Relazione Progetto
Advanced Programming Language

SkanCoin – criptovaluta

Alessandro Messina, matricola O55000354

Orazio Scavo, matricola O55000414

ANNO ACCADEMICO 2018/2019

Sommario

1 Introduzione

1.1 Obiettivi

L'obiettivo del progetto, così come indicato nel file di proposta dello stesso, è stato quello di realizzare una *criptovaluta* basata sulla tecnologia *blockchain distribuita* su una *rete p2p*. Ogni nodo della rete comprende:

- Un ***server HTTP* (REST)**
  - Espone le operazioni eseguibili da un utente sulla blockchain tramite interfaccia Web.
  - Composto da un backend realizzato in C++ e da un frontend realizzato con tecnologie Web quali HTML5, CSS3, Javascript e framework AngularJS.
- Un ***client e server p2p***
  - Tramite WebSocket consentono al nodo l'interazione con gli altri peer della rete per lo scambio delle informazioni necessarie alle operazioni svolte sulla blockchain.
  - Realizzato in C++.
- Un ***backend della criptovaluta***
  - Gestisce le operazioni sulla blockchain distribuita, sui wallet e sulle transaction pool dei nodi e viene utilizzato dal server HTTP e P2P.
  - Realizzato in C++.
- Un'***applicazione Web di diagnostica della blockchain***
  - Raccoglie informazioni sull'andamento della blockchain (numero di blocchi nel tempo, numero di transazioni nel tempo, etc.) e li mostra tramite dei grafici su browser.
  - Realizzata in R.

1.2 Architettura

L'architettura del sistema rispecchia quando indicato nel paragrafo 1.1. Un utente può usufruire dei servizi offerti dal sistema collegandosi all'indirizzo della Web Application e dell'applicazione di diagnostica R.

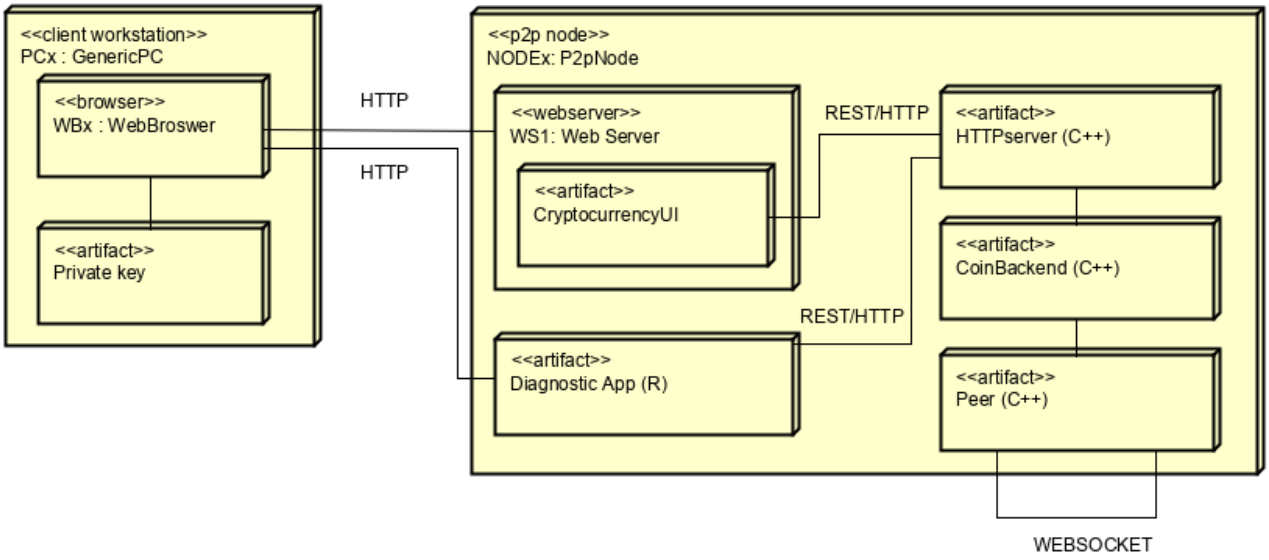

Figura 1. Architettura del sistema SkanCoin

1.3 Struttura del progetto

Il progetto si articola in tre elementi principali:

- ***SkanCoin***; contiene tutto il codice C++, dunque l'HTTP server, il coin backend (Blockchain) e il P2P server/client. SkanCoin è l'implementazione della criptovaluta e contiene tutti gli elementi necessari per il suo corretto funzionamento.

- ***Webapp***; la webapp utilizzata per accedere a SkanCoin.
- ***DiagnosticClient***; contiene il codice R per la visualizzazione delle statistiche della criptovaluta.

1.4 Suddivisione del lavoro tra gli studenti

Il sistema è stato progettato da entrambi gli studenti in ogni sua parte. L'implementazione è avvenuta invece con una suddivisione dei compiti così come riportato a seguire:

- Orazio Scavo. Implementazione dei file *Transactions.cpp, Transactions.hpp, Wallet.cpp, Wallet.hpp, Peer.cpp* e *Peer.hpp*
- Alessandro Messina. Implementazione dei file *HttpServer.cpp, HttpServer.hpp, config.hpp.in, DiagnosticClient* (*app.R* e *main.R*) e *WebApp (index.html, main.css* e *mainController.js*).

I restanti file sono stati implementati da entrambi gli studenti.

L'intera attività di sviluppo è stata effettuata su ambiente Linux (Ubuntu 18.04 e ArchLinux) utilizzando IDE e sistema di versioning GitHub (https://github.com/Taletex) al fine di consentire un migliore coordinamento tra i membri del gruppo di sviluppo.

2 Scelte implementative

A seguire sono riportate le scelte implementative adottate per ognuno degli elementi costituenti il sistema.

2.1 SkanCoin

SkanCoin è l'implementazione della criptovaluta basata su blockchain distribuita. La progettazione e lo sviluppo di SkanCoin è stato effettuato, dopo uno studio della tecnologia in questione, tramite C++, le sue librerie standard e delle librerie esterne necessarie per l'implementazione di alcuni elementi costituenti:

- *Crow* (https://github.com/ipkn/crow), usata per l'implementazione del server HTTP e del server P2P (WebSocket server).
- *Easywsclient* (https://github.com/dhbaird/easywsclient), usato per l'implementazione del client P2P (WebSocket client).
- *Rapidjson* (https://github.com/Tencent/rapidjson). Usato per il parsing del json nelle richieste POST HTTP nel server HTTP e per il parsing del json negli scambi di messaggi nelle WebSocket dei Peer.
- *Easy-ecc* (https://github.com/esxgx/easy-ecc). Usato per la cifratura asimmetrica (chiave pubblica e privata dei wallet degli utenti, gestione delle transazioni, etc..)
- *PicoSHA2* (https://github.com/okdshin/PicoSHA2). Usato per la cifratura SHA256 (generazione dell'hash dei blocchi della blockchain e dell'id delle transazioni).

Il progetto C++ è stato gestito tramite *Cmake* per quanto riguarda compilazione e linking dei file. Cmake è stato utilizzato anche per la gestione di un file di configurazione del progetto (config.hpp.in): tramite Cmake vengono passati i valori per alcune variabili di progetto, quali il nome del progetto, il numero di versione e un flag per la visualizzazione di informazioni di debug. Questo flag (DEBUG_INFO) viene utilizzato dal preprocessore C++ per inserire (a seconda che sia abilitato o meno) delle righe di codice all'inizio di ogni funzione per la stampa del nome della funzione, file e numero di riga di codice in cui la funzione si trova. Questa feature è stata inizialmente usata al posto di un vero debugger per provare le funzionalità del preprocessore con ottimi risultati. Si è deciso di continuare a mantenerla anche in seguito, nonostante diminuisca la leggibilità del codice, (all'inizio di ogni funzione tre righe sono interamente utilizzate per implementare questa feature) per fini didattici.

All'interno del progetto è stato infine fatto largo uso di classi, eccezioni, librerie standard (contenitori, output su console, output e input su file, gestione del tempo, verifica dei tipi, etc.), modificatori, e altri elementi della programmazione orientata agli oggetti forniti dal linguaggio C++.

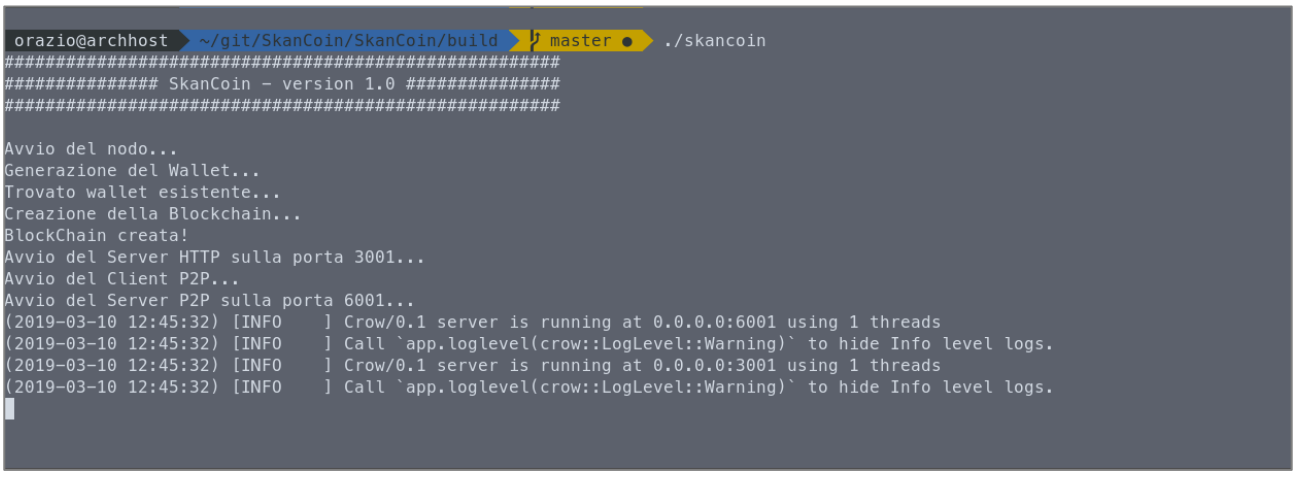

Figura 2. Output su terminale al momento di avvio del backend SkanCoin C++

2.1.1 Blockchain

La blockchain è la classe principale di tutto il progetto ed è stata implementata come *singleton* nei file Blockchain.cpp e Blockchain.hpp. Questa scelta è stata effettuata considerando che deve esistere una unica istanza della blockchain nel nodo (con il proprio stato e funzioni che agiscono su di esso). La vera e propria struttura blockchain è stata implementata dentro la classe Blockchain come una lista di Block poiché le operazioni più frequenti su di essa rappresentano inserimenti e rimozioni.

L'implementazione della blockchain è stata effettuata seguendone i relativi principi base, quali: generazione del blocco di genesi, generazione dei nuovi blocchi, calcolo degli hash dei nuovi blocchi tramite funzioni crittografiche di hashing (SHA256), validazione e integrità dei nuovi blocchi e della blockchain, replacement della blockchain sulla base della difficoltà cumulativa, proof-of-work nel mining dei blocchi (trovare il giusto hash del blocco che inizia con un certo numeri di zero, numero definito dalla difficoltà del blocco), gestione della difficoltà di mining dei blocchi (ovvero quando incrementare e quando decrementare la difficoltà di mining), interazione con gli altri nodi (tramite il P2P server/client) e interazione con l'utente (tramite http server). Sulla base della blockchain è stata poi definita la criptovaluta.

2.1.2 Criptovaluta

A partire dalla blockchain è stata implementata la criptovaluta. Questa si serve della struttura blockchain per lo scambio e la memorizzazione delle "monete virtuali". In particolare, le informazioni relative al proprietario e alla quantità di monete possedute sono situate dentro le transazioni presenti nei dati di ogni blocco della blockchain. Il codice della criptovaluta è situato nei file della blockchain, dei transaction component, delle transazioni, della transaction pool e del wallet. Parte della gestione avviene anche dentro il P2P client/server (parte relativa alla comunicazione tra i peer) e l'HTTP Server (parte relativa all'interazione dell'utente).

L'implementazione di una criptomoneta prevede che i dati contenuti nei blocchi della blockchain siano delle transazioni. La struttura delle transazioni viene definita all'interno del file Transaction.hpp. Questo file contiene anche una serie di funzioni utilizzate dalle altre parti dell'applicazione per gestire tutto ciò che riguarda i movimenti dei coin da un wallet all'altro. Il movimento dei coin implica la creazione e la validazione delle transazioni, validazione intesa sia in termini strutturali (struttura del dato e validità degli hash per evitare modifiche), che in termini logici, infatti all'interno di questo file è implementato anche il controllo degli output di transazione che non sono stati utilizzati come input di altre e che quindi rappresentano i coin effettivamente "posseduti" dai vari utenti (wallet). Le funzioni implementate all'interno di questo file non sono state associate ad una classe in quanto intese come un insieme di funzionalità generiche, non associate ad una specifica istanza.

Un altro concetto molto importante delle criptovalute riguarda la possibilità di inviare delle transazioni agli altri peer della rete in modo da consentire ad un utente di non dover effettuare personalmente il mining di un nuovo blocco contenente delle transazioni. Questo concetto è stato realizzato tramite la cosiddetta *transaction pool*: questa, implementata come singleton nei file TransactionPool.cpp e TransactionPool.hpp, mantiene una lista di tutte le transazioni inviate ai peer della rete e non ancora aggiunte ad un blocco della blockchain. Quando un utente invia una nuova transazione nella transaction pool del nodo, viene effettuato un broadcast dell'informazione a tutti i peer così che anch'essi possano aggiungere la transazione alla loro transaction pool. In questo modo, ogni peer può decidere spontaneamente di effettuare il mining di un nuovo blocco utilizzando come dati le transazioni della propria transaction pool.

Infine, un aspetto fondamentale del funzionamento delle criptomonete è rappresentato dall'uso di meccanismi di cifratura asimmetrica, che permettono agli utenti di possedere una certa quantità di coin ed effettuare delle transazioni in maniera sicura ed anonima. Tutto ciò che riguarda i meccanismi di sicurezza della blockchain, ossia le procedure di firma digitale applicate agli input dei vari blocchi, è stato implementato all'interno dei file Wallet.hpp e Wallet.cpp. In particolare, sono state implementate le procedure di firma e relativa verifica sugli input di transazione utilizzando l'algoritmo ECDSA (basato sulle curve ellittiche). Questo

file si occupa della creazione e della memorizzazione della coppia di chiavi associate al nodo, infatti queste vengono conservate all'interno di appositi file. L'utilizzo delle funzioni di libreria per la generazione delle chiavi e delle firme e per la relativa verifica viene affiancato a delle specifiche funzioni di conversione da array di byte a stringhe e viceversa, in modo da poter effettuare la manipolazione di tali chiavi in formato stringa in tutto il resto dell'applicazione, ottenendo inoltre un formato stampabile e leggibile per le chiavi pubbliche, che rappresentano gli indirizzi dei wallet degli utenti a cui queste appartengono. Il formato adottato prevede la stampa dei valori numerici corrispondenti ai singoli byte, separati da punti (Esempio: "124.65.87.23.1.65..."). Per gli stessi motivi descritti per il file Transaction.cpp, il file Wallet non contiene l'implementazione dei metodi di una classe, ma semplicemente un insieme di funzioni generiche che possono essere utilizzate dalle altre parti dell'applicazione.

2.1.3 Server http

Il server HTTP consente ad un utente di utilizzare i meccanismi della criptovaluta. Il server HTTP fornisce un'API REST che agisce sul sistema SkanCoin e che viene utilizzata dalla webapp per una più facile fruizione del sistema da parte dell'utente. Per poter realizzare il server HTTP, come già detto, sono state utilizzate le librerie Crow e Rapidjson.

- La *libreria Crow* è stata utilizzata per definire il servizio REST in maniera leggibile e agile. Per gestire meglio le risposte da fornire al client del servizio REST si è deciso di definire delle risposte standard (oggetto Response di Crow) nel quale viene indicato il content-type di ritorno e l'Access-Control-Allow-Origin per gestire il CORS (Cross-Origin Resource Sharing, Crow non gestisce il CORS automaticamente, è stato quindi necessario farlo esplicitamente nel codice).
- La *libreria RapidJson* è stata utilizzata per effettuare il parsing dei body delle richieste POST al servizio REST.

Si è scelto di implementare le seguenti richieste REST per poter agire sulla criptovaluta SkanCoin:

- GET: webresources/publickey. Ritorna la chiave pubblica dell'utente.
- GET: webresources/blocks. Ritorna la blockchain.
- GET: webresources/blocks/blockId. Ritorna un blocco, dato il suo hash.
- GET: webresources/transactions/transactionId. Ritorna una transazione dato il suo id.
- GET: webresources/unspentTransactionOutputs. Ritorna gli output non spesi dell'intera blockchain. Questi conterranno la quantità totale di monete disponibili nei wallet di tutta la blockchain.
- GET: webresources/unspentTransactionOutputs/address. Ritorna gli output non spesi appartenenti ad un certo indirizzo (wallet).
- GET: webresources/myUnspentTransactionOutputs. Ritorna gli output non spesi relativi al wallet dell'utente corrente.
- GET: webresources/balance. Ritorna il bilancio del wallet (quantità totale di monete disponibili nel wallet).
- GET: webresources/transactionPool. Ritorna la transaction pool del nodo.
- POST: webresources/transactions. Crea una nuova transazione e la inserisce nella transaction pool del nodo. Viene effettuato anche il broadcast a tutti gli altri peer della rete in modo che possano aggiornare la loro transaction pool.
- POST: webresources/blocks/pool. Effettua il mining di un nuovo blocco utilizzando le transazioni del transaction pool (più la coinbase transaction).
- POST: webresources/blocks/transactions. Effettua il mining di un nuovo blocco contenente la coinbase transaction e una transazione con uno o più output destinazione.
- POST: webresources/peers. Aggiunge un peer alla lista di peer.
- GET: webresources/peers. Ritorna il numero di peer della rete.
- GET: webresources/stats/filename. Ritorna il contenuto del file di nome filename. Usato per la raccolta delle statistiche da parte dall'applicazione R.

2.1.4 Client e server P2P

La natura distribuita della tecnologia delle blockchain richiede che ci siano diversi nodi paritetici connessi in rete, in modo da mantenere una versione comune della blockchain che diventa valida nel momento in cui questa viene approvata dalla maggior parte dei nodi connessi, senza avere nessun punto di controllo centralizzato. Il file Peer.cpp implementa in una classe Singleton la logica di gestione delle connessioni e dei messaggi in arrivo da altri peer, invocando le funzionalità della blockchain definite nelle altre parti dell'applicazione. Per modellare i messaggi scambiati è stata utilizzata una classe definita appositamente, Message, che presenta un metodo tostring per ottenere una stringa contenente la rappresentazione del messaggio in formato JSON. Per il parsing dei messaggi è stata utilizzata anche in questo caso la libreria "RapidJson".

L'implementazione del peer ha presentato delle particolari criticità in quanto non è stato possibile utilizzare la stessa libreria per la gestione delle socket aperte dal nodo (parte client del peer) e quelle create a partire da delle connessioni in arrivo da altri nodi (parte server). Questo ha determinato la presenza di socket di due tipi diversi da gestire. Le librerie utilizzate sono state Crow per la parte server ed Easywsclient per la parte client. La prima prevede una gestione asincrona dei messaggi in arrivo al server peer (si ha la possibilità di definire degli handler per le nuove connessioni, per i messaggi in arrivo e per la chiusura delle socket da parte dei client), mentre la seconda richiede un polling sulle socket client per verificare che ci siano nuovi messaggi in arrivo o socket che sono state chiuse. Vista la natura differente delle due gestioni è stato necessario definire due thread diversi che gestiscono i due tipi di socket. La complicazione principale è data dal fatto che sia il thread client che quello server, così come il thread associato al server http, hanno la necessità di effettuare dei broadcast su tutte le socket aperte per il nodo, a prescindere che queste siano state aperte dal thread client o da quello server. L'accesso alle socket da parte di thread diversi ha dunque richiesto l'uso di un mutex per coordinare l'accesso ai riferimenti dei due diversi tipi.

Altra nota importante è data dalla necessità di rispettare certe interfacce di definizione per le funzioni usate come handler per i messaggi in arrivo al peer client, è stato infatti necessario definire tra i membri della classe peer un reference di tipo easywsclient::socket, in modo da avere un riferimento alla socket corrente all'interno dell'handler, in quanto non si ha un riferimento fisso dato che il polling viene effettuato su una lista non ordinata di socket client. Per permettere l'utilizzo dello stesso metodo per gestire i messaggi in arrivo ai due thread, lo stesso pattern è stato usato anche per il server Peer.

2.2 DiagnosticClient

Il DiagnosticClient è l'applicazione R per la gestione e visualizzazione delle statistiche della blockchain (numero di blocchi, tranazioni e coin della blockchain nel tempo, tempo di mining per ogni blocco e tempo di attesa per la conferma di ogni transazione).

2.2.1 Package Shiny

Una prima scelta implementativa è stata quella di servirsi del package *Shiny* per la realizzazione di un'applicazione web R. Tramite Shiny è possibile hostare sul web l'applicazione R, rendendo più facile il suo utilizzo da parte di un utente finale che deve così semplicemente collegarsi all'indirizzo sul quale l'applicazione stessa è hostata. Così facendo non è necessario che il client possegga l'applicazione R, poiché è il server a hostarla.

Per realizzare l'applicazione R con Shiny è stato necessario definire l'interfaccia grafica (Fig. 3) e il server contenente la logica applicativa. L'interazione tra le due parti consiste nell'uso del pattern observer su alcune variabili.

2.2.2 Raccolta e visualizzazione dei dati

Per raccogliere i dati da visualizzare vengono effettuate delle chiamate HTTP verso l'HTTP server utilizzando il package *jsonlite*. Le richieste vengono effettuate quando l'utente clicca sugli appositi pulsanti dell'interfaccia grafica (tre possibili richieste). I dati ritornati da queste richieste sono elaborati sotto forma di data frame e visualizzati utilizzando le funzionalità del package *ggplot2*.

Si è deciso di visualizzare le statistiche *numero di blocchi nel tempo, numero di transazioni nel tempo* e *numero di coin nel tempo* della blockchain in un unico grafico in modo da poter rendere meglio l'idea di come la blockchain evolve. Il grafico scelto per la visualizzazione di tali statistiche è un semplice grafico time-series (valore-tempo). Lato backend, i dati per le statistiche sono memorizzati in un file di log (blockchainstats.txt) aggiornato ad ogni inserimento di un nuovo blocco o sostituzione della blockchain.

Per quanto riguarda il *tempo di mining di ogni blocco* si è decisa una rappresentazione tramite diagramma a barre (tempo – indice blocco). Lato backend, i dati necessari per questa statistica sono memorizzati anch'essi in un file di log (blocksminingtime.txt) che viene aggiornato ogni qual volta viene effettuato il mining di un nuovo blocco. In tale occasione, durante il broadcast delle informazioni sul nuovo blocco, vengono inviate anche le informazioni riguardo il suo tempo di mining in modo che chiunque decida di aggiungere tale blocco alla propria blockchain possa anche aggiornare il file di log sui tempi di mining dei vari blocchi così da fornire una view coerente con quella di tutti gli altri nodi.

Infine, anche il *tempo di attesa per la conferma di ogni transazione* è rappresentato tramite un diagramma a barre (tempo – id transazione). La raccolta dei dati per questa statistica è un po' più complessa delle altre e coinvolge nuovamente un file di log (transactionwaitingtime.txt). Ogni qual volta viene aggiunta una transazione nella transaction pool di un nodo viene memorizzata anche l'informazione relativa al tempo di inserimento. Ogni qual volta una transazione è prelevata dalla transaction pool per effettuarne il mining in un blocco, si sfrutta l'informazione relativa al suo tempo di inserimento nel pool per determinare il tempo di attesa all'interno della transaction pool stessa. Dunque, si aggiunge una riga all'interno del file di log e si manda in broadcast l'informazione così che tutti i nodi possano aggiornare i loro file di log. Così facendo la view della statistica sarà uguale a prescindere dal nodo contattato.

È bene notare che il formato di salvataggio di ogni statistica è lo stesso per tutte e tre i gruppi di statistiche: ogni elemento è memorizzato su una riga diversa del relativo file di log come se fosse un oggetto JSON. In questo modo l'HTTP nel momento in cui ha bisogno di leggere le statistiche si può limitare a prelevare una riga alla volta del corretto file di log e inserirla in un array JSON da ritornare verso il client (R).

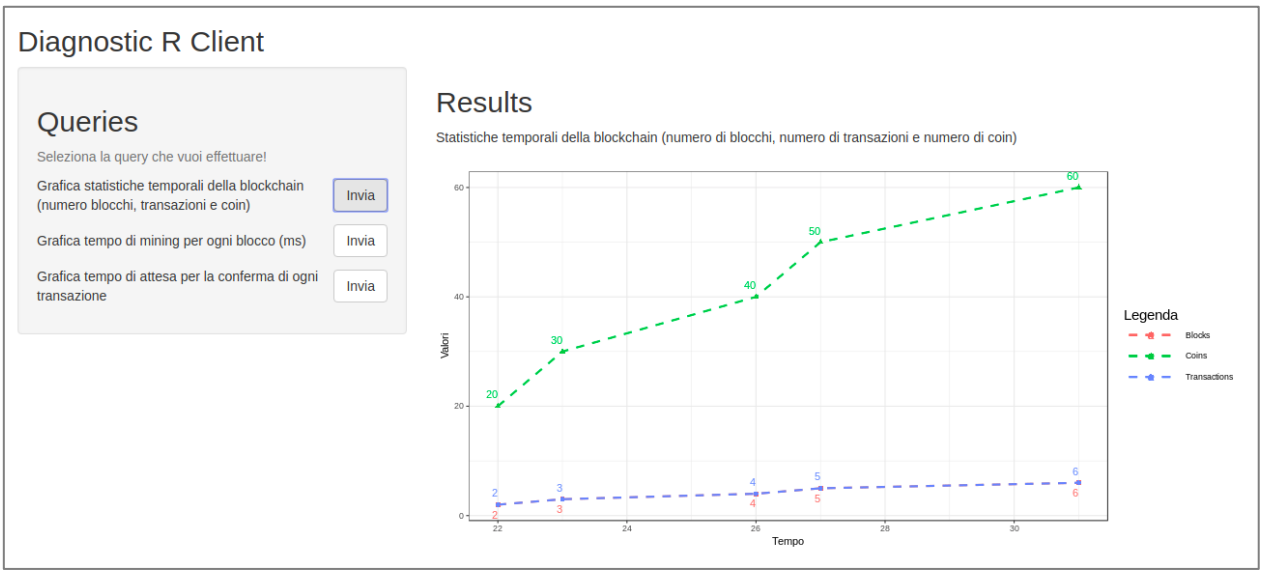

Figura 3. Interfaccia grafica dell'applicazione R per la raccolta e visualizzazione delle statistiche della criptovaluta.

2.3 WebApp

L'applicazione web è stata realizzata per consentire un più facile utilizzo del sistema da parte di un utente che non conosce la sua implementazione interna. Per poter utilizzare il sistema, l'utente deve così collegarsi solamente all'indirizzo della web application e servirsi della sua interfaccia grafica (Fig. 4). In questa interfaccia è mostrata la chiave pubblica dell'utente e una serie di query che esso può svolgere sul sistema, ad esempio la stampa dell'intera blockchain o l'invio di una nuova transazione nel transaction pool. Una query consiste in una chiamata al servizio REST fornito dal server HTTP in esecuzione sul nodo a cui si ci è collegati. La risposta del server HTTP può essere visualizzata nella sezione "Query output" dell'interfaccia grafica.

Per quanto riguarda la WebApp non sono state effettuate scelte implementative importanti (se non quella di realizzare la web app stessa).

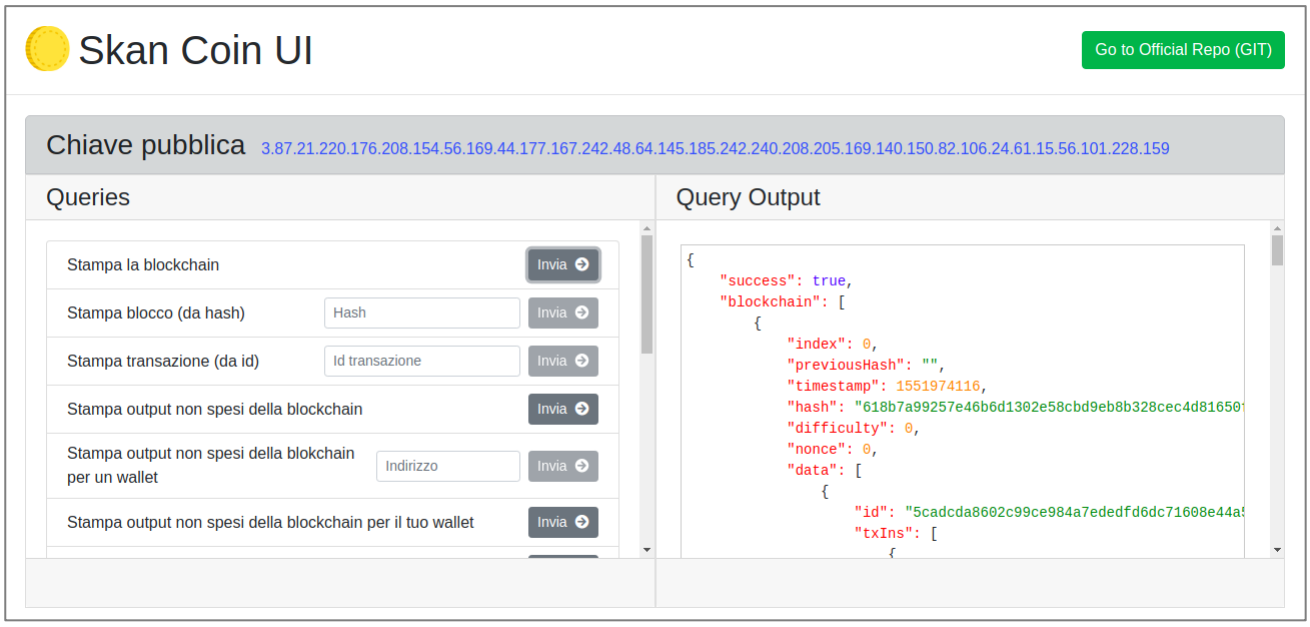

Figura 4. Interfaccia grafica della web application.

3 File readme: requisiti, dipendenze e avvio del progetto

All'indirizzo https://github.com/Taletex è disponibile il file readme nel quale è possibile trovare:

- **Requisiti dell'applicazione (requirements)**. Sono indicati i requisiti necessari per far avviare l'applicazione e le operazioni da svolgere per poterli installare. Le indicazioni riguardano un sistema operativo Ubuntu 18.04.
- **Dipendenze (dependencies)**. Sono indicate tutte le librerie e package utilizzati nel sistema.
- **Istruzioni per l'esecuzione dell'applicazione (Running for test)**. Contiene l'insieme di operazioni da effettuare per poter eseguire il sistema una volta soddisfatti i suoi requisiti (requirements).