**Relazione Progetto**

**Advanced Programming Language**

**SkanCoin – criptovaluta**

**Alessandro Messina, matricola O55000354**

**Orazio Scavo, matricola O55000414**

ANNO ACCADEMICO 2018/2019

Sommario

[***1*** ***Introduzione*** - 2 -](#_Toc2963373)

[**1.1** **Obiettivi** - 2 -](#_Toc2963374)

[**1.2** **Architettura** - 2 -](#_Toc2963375)

[**1.3** **Struttura del progetto** - 2 -](#_Toc2963376)

[**1.4** **Suddivisione del lavoro tra gli studenti** - 3 -](#_Toc2963377)

[***2*** ***Scelte implementative*** - 4 -](#_Toc2963378)

[**2.1** **SkanCoin** - 4 -](#_Toc2963379)

[**2.1.1** **Blockchain** - 5 -](#_Toc2963380)

[**2.1.2** **Criptovaluta** - 5 -](#_Toc2963381)

[**2.1.3** **Server http** - 5 -](#_Toc2963382)

[**2.1.4** **Client e server P2P** - 6 -](#_Toc2963383)

[**2.2** **DiagnosticClient** - 6 -](#_Toc2963384)

[**2.2.1** **Package Shiny** - 6 -](#_Toc2963385)

[**2.2.2** **Raccolta e visualizzazione dei dati** - 7 -](#_Toc2963386)

[**2.3** **WebApp** - 8 -](#_Toc2963387)

[***3*** ***File readme: requisiti, dipendenze e avvio del progetto*** - 10 -](#_Toc2963388)

# ***Introduzione***

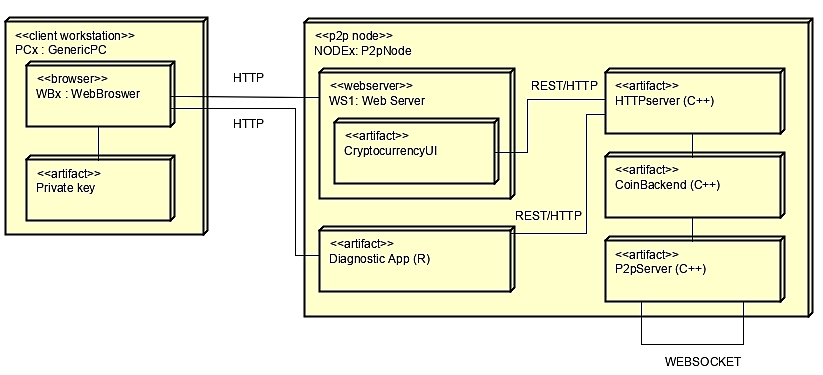
## **Obiettivi**

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.

## **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.



## **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.

## **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.

# ***Scelte implementative***

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

## **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.
* *Easywsclient* (<https://github.com/dhbaird/easywsclient>), usato per l’implementazione del clientP2P.
* *Rapidjson* (<https://github.com/Tencent/rapidjson>). Usato per il parsing del json nelle richieste POST HTTP nel server HTTP.
* *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) a 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++.

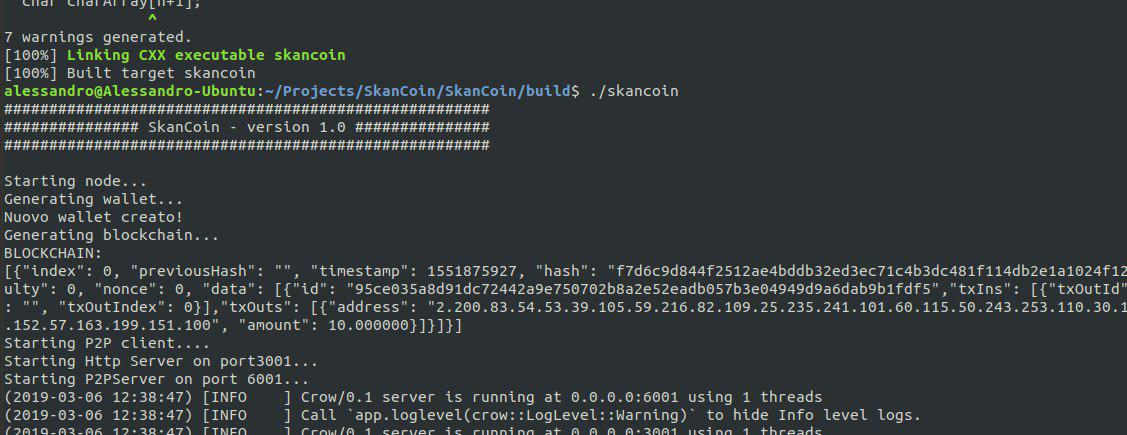


Figura 1. Output su terminale al momento di avvio del backend SkanCoin C++

### **Blockchain**

La blockchain è la classe principale di tutto il progetto ed è stata implementata come *singleton*. 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 genesis, 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.

### **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 sia nei file della blockchain, sia in quelli delle transazioni, transaction pool e wallet. Parte della gestione avviene anche dentro il P2P client/server e l’HTTP Server.

Parlare del file Transactions.cpp e .hpp (in maniera generica così come ho fatto blockchain). Qui o in wallet bisogna parlare dell’uso della cifratura asimmetrica e della scelta di convertire le chiavi da array di byte a stringa.

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 implementato 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, ma 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 della rete 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.

Parlare del file Wallet.cpp e .hpp (sempre in maniera generica)

### **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 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 molto 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.

### **Client e server P2P**

* Breve descrizione
* Parlare di crow, easywsclient e del problema della sincronizzazione sulle risorse condivise (il fatto che hai dovuto usare i mutex).

## **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).

### **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. 2) e il server contenente la logica applicativa. L’interazione tra le due parti consiste nell’uso del pattern observer su alcune variabili.

### **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 dell’intera 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 qualunque nodo sia interrogato si avrà sempre una unica view circa questa statistica.

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).

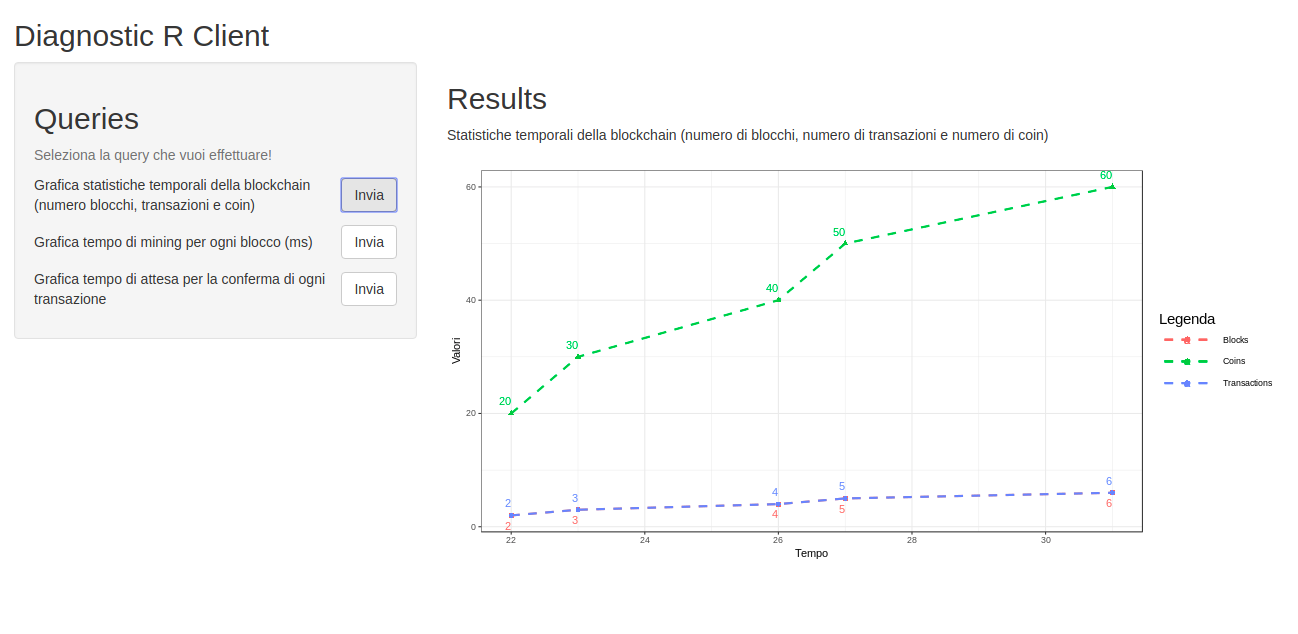


Figura 2. Interfaccia grafica dell'applicazione R per la raccolta e visualizzazione delle statistiche della criptovaluta.

## **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. 3). 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).

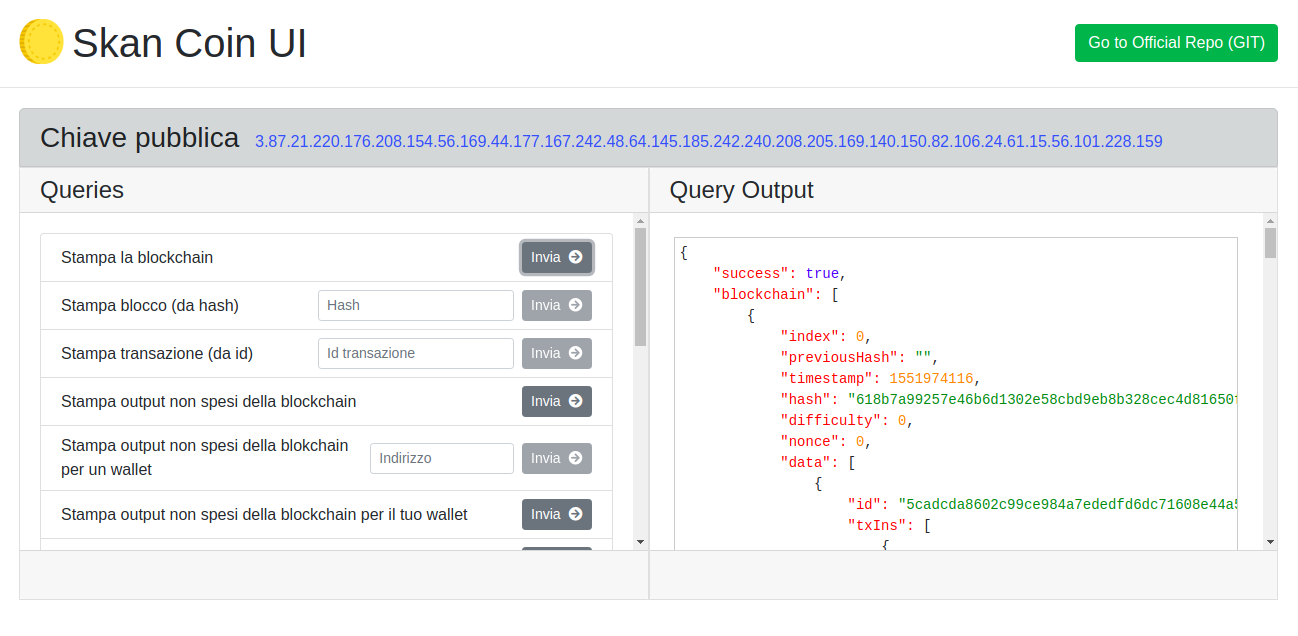


Figura 3. Interfaccia grafica della web application.

# ***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).