**Relazione Progetto**

**Advanced Programming Language**

**SkanCoin – criptovaluta**

**Alessandro Messina, matricola O55000354**

**Orazio Scavo, matricola O55000414**

ANNO ACCADEMICO 2018/2019

Sommario

[***1*** ***Introduzione*** - 2 -](#_Toc2896706)

[**1.1** **Obiettivi** - 2 -](#_Toc2896707)

[**1.2** **Architettura** - 2 -](#_Toc2896708)

[**1.3** **Struttura del progetto** - 2 -](#_Toc2896709)

[**1.4** **Suddivisione del lavoro tra gli studenti** - 3 -](#_Toc2896710)

[***2*** ***Scelte implementative*** - 4 -](#_Toc2896711)

[**2.1** **SkanCoin** - 4 -](#_Toc2896712)

[**2.2** **DiagnosticClient** - 4 -](#_Toc2896713)

[**2.2.1** **Package Shiny** - 4 -](#_Toc2896714)

[**2.2.2** **Raccolta e visualizzazione dei dati** - 4 -](#_Toc2896715)

[**2.3** **WebApp** - 5 -](#_Toc2896716)

[***3*** ***File readme: requisiti, dipendenze e avvio del progetto*** - 6 -](#_Toc2896717)

# ***Introduzione***

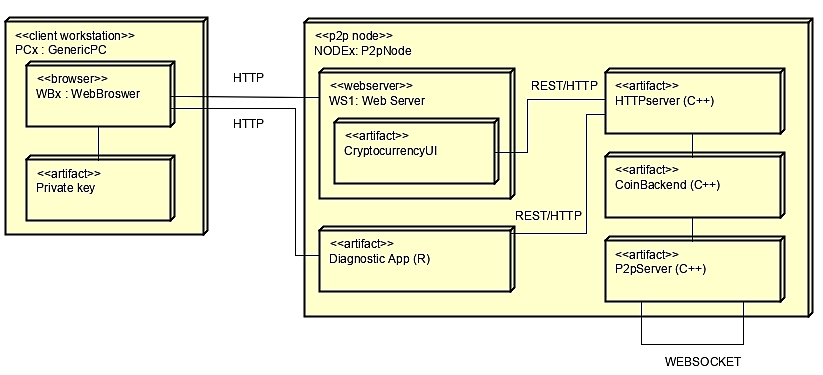
## **Obiettivi**

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**

Lorem Ipsum.

### **Blockchain**

Blockchain decription.

### **Criptovaluta**

Crypto description.

### **Server HTTP**

http server description

### **Client e server P2P**

P2P server description.

## **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 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).

## **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. 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).

# ***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).