ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

# REALIZZAZIONE DI UNA BLOCKCHAIN PRIVATA PER LA GESTIONE DI UNA RETE DI PAGAMENTI

*Elaborato in*
SISTEMI DISTRIBUITI

*Relatore*
Prof. ANDREA OMICINI

*Presentata da*
ELIA PASQUALINI

Anno Accademico 2020 – 2021

# PAROLE CHIAVE

Blockchain

Stellar

Consenso

Token

*Ai miei genitori,*
*per aver sempre creduto in me*

# Indice

# Elenco delle figure

# Introduzione

A distanza di quattordici anni dalla sua nascita la *blockchain* rappresenta oggi un paradigma e una piattaforma di innovazione che permette di dare nuove risposte a tanti e diversi bisogni di imprese, organizzazioni, cittadini e consumatori.
La blockchain, così come è accaduto per Internet, sta entrando gradualmente nelle nostre vite, spesso in modo indiretto, in molti casi solo come possibile soluzione a piattaforme che risolvono in modo innovativo i nostri bisogni o i bisogni delle imprese nelle quali lavoriamo. Quale che sia la modalità con cui veniamo in contatto con la blockchain appare sempre più importante conoscerla, avere contezza delle sue prospettive e delle sue potenzialità.
In particolare, a differenza di altre innovazioni tecnologiche, la blockchain fa riferimento ad alcuni temi e concetti apparentemente molto diversi e lontani tra loro, che normalmente non vengono associate all'innovazione digitale: la fiducia, la responsabilità, la comunità, la decentralizzazione. In più, accanto a questi ce ne sono altri che hanno una forte relazione con la tecnologia, ma che a loro volta non sono "consueti" come ad esempio i temi della trasparenza, dell'immutabilità, della condivisione e della competizione nel raggiungimento di un risultato.
Nel seguente documento di tesi lo scopo della tecnologia blockchain è quello di creare un'architettura sicura per gli scambi di *utility token* basata su un sistema distribuito, ovvero un insieme eterogeneo formato da più calcolatori che appare all'utilizzatore come un unico dispositivo.
Nel capitolo 1 verrà analizzato lo stato dell'arte del mondo blockchain. Partendo dalla descrizione della tecnologia basata sui registri distribuiti sarà analizzato il funzionamento della blockchain stessa, le sue caratteristiche e la struttura che la compone, elencandone le varie tipologie. Successivamente, con partico-

lare attenzione, verrà analizzata l'architettura della rete Stellar ed il proprio protocollo per il raggiungimento del consenso.
Nel capitolo 2 saranno presentati con maggior dettaglio gli obiettivi di questo progetto ed anche tutti i requisiti che il sistema dovrà soddisfare.
Attraverso l'uso di diagrammi saranno delineate le funzionalità del sistema e le entità coinvolte.
Nel capitolo 3 verrà descritto in che modo è stata progettata la rete evidenziando in particolar modo i requisiti hardware, la definizione di API e la configurazione dei server. In seguito ai test effettuati saranno proposte 3 soluzioni con le quali, in base ai requisiti di sicurezza e tolleranza ai guasti, sarà realizzata l'infrastruttura di rete.
Per chiarire al meglio il lavoro svolto, nel capitolo 4, saranno proposti esempi di codice utilizzati per la configurazione dei nodi Stellar e per l'implementazione degli *handler* delle API. Il capitolo 5 introdurrà le possibili modalità con cui si potrà decidere di espandere il progetto in futuro.
Nel capitolo 6 saranno presentati i vantaggi derivanti dall'uso della tecnologia blockchain in ambito finanziario. Come conclusione verranno proposte le considerazioni riguardo tutto il progetto di tesi.

# Capitolo 1

# Stato dell'arte

## 1.1 Distributed Ledger Technology

La *Distributed Ledger Technology (DLT)* fa riferimento ai processi e alle tecnologie correlate che consentono ai nodi di una rete di proporre, convalidare e registrare in modo sicuro modifiche di stato (o aggiornamenti) ad un *synchronised ledger* distribuito tra i nodi della rete.
Per comprendere il funzionamento della Distributed Ledger Technology bisogna prima chiarire il significato della parola *ledger*: libro mastro. Il libro mastro è alla base della contabilità, ed è da sempre utilizzato per gestire scambi e relazioni commerciali. Esso rappresenta una memoria storica e serve in quanto può essere consultato in ogni momento per verificare le transazioni effettuate. Dai libri mastri in carta sono stati fatti incredibili passi avanti, ma il principio resta il medesimo.
Gli stessi libri mastri sono nati seguendo una logica centralizzata (*central authority*): un addetto si occupava del *data entry* e qualcuno li gestiva o ne estraeva i dati. Una logica rimasta immutata anche nella prima traduzione digitale di questo strumento. La Distributed Ledger Technology permette, grazie ad elevati standard di sicurezza, di compiere un passaggio ulteriore, approdando a una logica distributiva.
La Distributed Ledger Technology è una tecnologia che può limitare il rischio di frodi, proprio grazie all'alto livello di sicurezza offerto dagli algoritmi e dalla crittografia. Un altro elemento di garanzia consiste nel fatto che i cambiamenti

apportati nel registro sono visibili ai membri della rete. Nei mercati finanziari, la DLT potrebbe eliminare i ruoli di mediazione, mutandone profondamente il funzionamento. Questa tecnologia è sicura ed economica e consente l'implementazione di servizi scalabili a livello globale, è a prova di manomissione ed è verificabile.
Tra i maggiori vantaggi che derivano dal'uso della DLT troviamo:

- **trasparenza e fiducia** : rende le interazioni e le transazioni comprensibili, tracciabili, certificabili e responsabili, particolarmente utile e rilevante per le interazioni tra utenti non attendibili.
- **sicurezza** : gestisce la crittografia dei dati, il controllo degli accessi, i dati a prova di manomissione e la tolleranza ai guasti.
- **riduzione dei costi** : attraverso la disintermediazione aumenta l'efficienza.

## 1.2 Blockchain

La blockchain, che è tanto conosciuta al giorno d'oggi grazie al mondo delle criptovalute, è un modo per implementare un DLT, ma non tutti i registri distribuiti utilizzano necessariamente la blockchain.
Questa tecnologia fornisce un modello per il calcolo distribuito e decentralizzato che può essere implementata in diversi modi all'interno di domini applicativi eterogenei e con obiettivi differenti.
Le blockchain hanno trovato applicazione in numerosi campi, vengono ad esempio utilizzate da alcune aziende produttrici di beni alimentari per tracciare i singoli prodotti, e basandosi sull'immutabilità dell'informazione, permettono al cliente di sapere con precisione e senza rischi di alterazione la reale origine del prodotto.
Altri esempi di applicazione sono la partnership tra banche di vari paesi al fine di semplificare il commercio internazionale ed inoltre si valuta la possibilità di utilizzarla nel campo politico per esprimere il proprio voto.
La blockchain gestisce un registro condiviso, trasparente, distribuito, *append-only* cioè, un registro che tiene traccia dell'ordine e del tempo delle transazioni protetto mediante schemi crittografici.

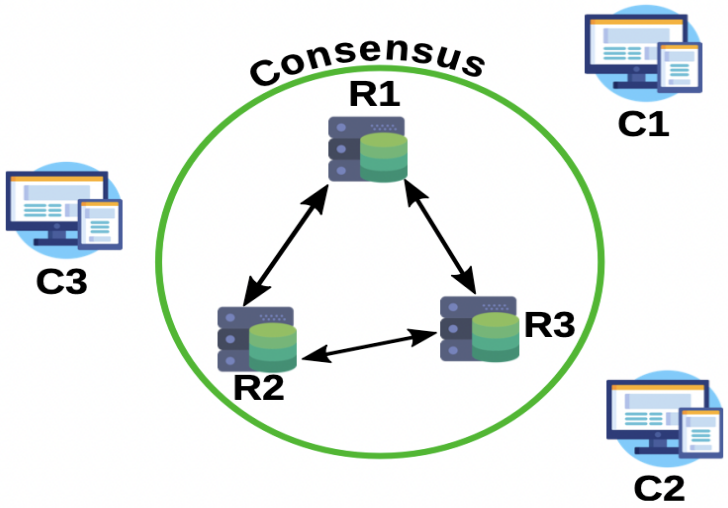


Figura 1.1: Rappresentazione architettura blockchain

Il registro, come mostrato in Figura 1.1, essendo replicato su più nodi di una rete *peer-to-peer* evita la necessità di un intermediario centralizzato fidato che rende il registro immutabile. Le transazioni descritte all'interno sono approvate e propagate attraverso il consenso, prevenendo la corruzione dei dati a causa di arresti anomali e/o menzogne delle macchine.
Il caso d'uso ideale di questa tecnologia è un dominio in cui le parti reciprocamente non si fidano ed hanno bisogno di interagire.
Nei sistemi classici è presente un soggetto, come un istituto bancario, che svolge il ruolo di garante. Esso garantisce a tutti i soggetti che vogliono effettuare transazioni che quest'ultime saranno svolte nel modo corretto. Esso diventa quindi un soggetto verso il quale c'è fiducia (*trusted*) da parte degli utenti, incidendo anche nella struttura che diviene centralizzata.
Nel modello della blockchain invece si vuole realizzare una struttura priva della figura del garante. Per fare ciò è necessario che la maggior parte dei soggetti sia trusted, e questo fa sì che la struttura sia di tipo peer-to-peer.

### Tipologie di Blockchain

Prima di soffermarsi sulle differenti tipologie di blockchain esistenti, è bene spiegare due importanti conflitti che questa tecnologia si trova ad affrontare. Il primo riguarda la contrapposizione della trasparenza con la privacy. Se è vero che una caratteristica distintiva della blockchain è il suo essere aperta e trasparente, la condivisione di informazioni mina la privacy dal momento in cui le transazioni sono accessibili pubblicamente da qualsiasi nodo facente parte della rete. Questa prima conflittualità può essere ricondotta alle operazioni di lettura della blockchain.
Il secondo conflitto si basa sulla antitesi tra sicurezza e velocità. La sicurezza è un concetto chiave della blockchain e per far sì che i dati siano salvati in maniera sicura è necessario che si utilizzino algoritmi di consenso che, il più delle volte, richiedono un significativo dispendio temporale. Questo contrasta con i requisiti di velocità e scalabilità in diverse applicazioni del contesto commerciale.
É possibile distinguere due principali tipi di blockchain che affrontano in maniere differenti i suddetti conflitti. Esse sono:

- Le *Permissionless Ledger* o blockchain pubbliche, di cui l'esempio più famoso e diffuso è rappresentato dalla Blockchain Bitcoin. Quest'ultime sono aperte, non hanno una "proprietà" o un attore di riferimento e sono concepite per non essere controllate. Il loro obiettivo è quello di permettere a ciascuno di contribuire all'aggiornamento dei dati sul ledger e di disporre, in qualità di partecipante, di tutte le copie immutabili di tutte le operazioni. Questo modello di blockchain impedisce ogni forma di censura, nessuno è nella condizione di impedire che una transazione sia aggiunta al ledger dopo che questa che ha conquistato il consenso necessario tra tutti i nodi (partecipanti) della blockchain.

- Le *Permissioned Ledger* o blockchain private, d'altra parte, sono sicuramente più vicine alle esigenze delle imprese essendo più sicure ed anche più performanti e veloci delle blockchain pubbliche. Queste possono infatti essere controllate e dunque avere una "proprietà". Quando un nuovo dato o record viene aggiunto, il sistema di approvazione non è vincolato alla maggioranza dei partecipanti della blockchain, bensì alla maggioranza di un numero limitato di attori che sono definibili come trusted. Le Permissioned Ledger permettono inoltre di definire speciali regole per l'accesso e la visibilità di tutti i dati. In altre parole introducono nella blockchain un concetto di *Governance* e di definizione di regole di comportamento.

**Funzionamento tecnologia**

La tecnologia blockchain, come sopra descritto, è un registro distribuito criptato contenente tutte le transazioni avvenute tra le parti partecipanti di un network.
Il registro è organizzato come una catena di blocchi, ognuno contenente una parte delle informazioni e queste sono memorizzate sequenzialmente in ordine temporale.
L'aggiunta di una qualsiasi transazione al registro richiede una verifica della sua validità da parte dei partecipanti al sistema. Una volta che la transazione è considerata verificata, viene inserita indelebilmente in un blocco della catena e quindi aggiunta in tutti i registri posseduti dai nodi. È possibile aggiungere

alla catena nuovi blocchi, ma non è possibile né modificare né eliminare blocchi precedenti. Viene cioè impedita ed evitata la modifica retroattiva. Una rappresentazione della struttura a blocchi connessi della blockchain è riportata in Figura 1.2.

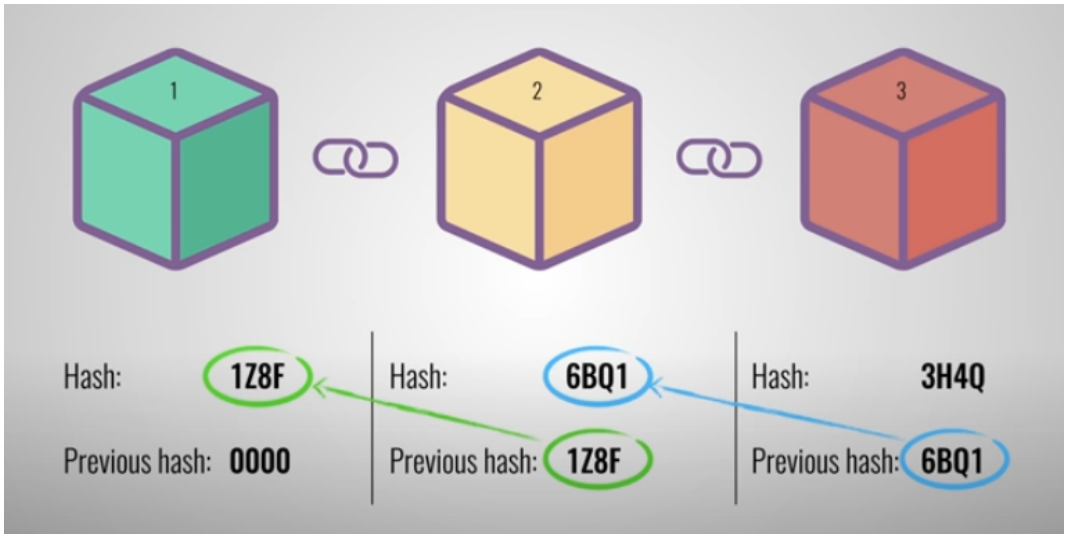


Figura 1.2: Rappresentazione blockchain e meccanismo di hash

Le transazioni possono essere ricondotte al primo blocco, comunemente chiamato il blocco della genesi che, come si può notare in Figura 1.2, è l'unico blocco che non è collegato ad un blocco precedente.
La blockchain contiene una registrazione certa e verificabile di ogni singola transazione effettuata tra i vari blocchi. L'immutabilità della blockchain è legata al fatto che ogni valore di hash è calcolato sulla base del contenuto del blocco e dell'hash del blocco precedente, come mostrato in Figura 1.2.

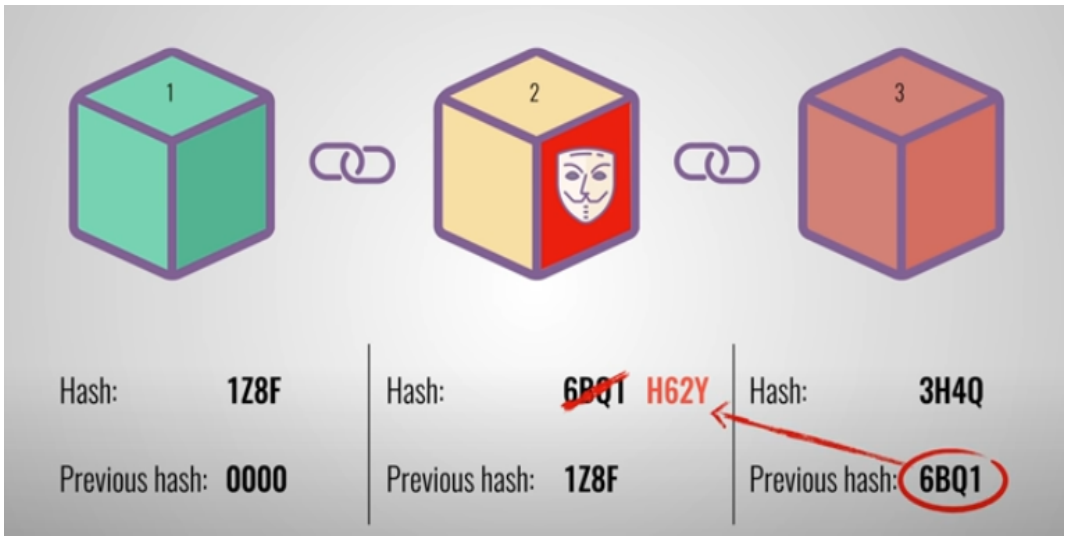


Figura 1.3: Rappresentazione immutabilità blockchain

Per tale ragione, se un utente volesse manomettere il blocco **i**, dovrebbe ricalcolare l'hash del blocco **i** come si può vedere in Figura 1.3. Questo richiederà a sua volta di ricalcolare l'hash di tutti i blocchi successivi, poiché la modifica di un hash invaliderà tutti gli hash successivi, essendo ogni valore strettamente collegato ai precedenti. Poichè il libro mastro è distribuito su tutti i nodi, sarebbe quindi necessario cambiare l'hash di almeno il 51% dell'intera rete, e tale enorme sforzo richiesto in termini computazionali rende tale operazione quasi impossibile nella pratica.

## 1.3 Stellar

Per la realizzazione del seguente progetto è stata scelta la tecnologia Stellar. Stellar è una rete open-source per le valute e i pagamenti. La rete Stellar ha una valuta digitale nativa, il *lumen* (XML), che viene richiesta in piccole quantità per inizializzare i conti ed effettuare transazioni. Stellar non privilegia alcuna valuta in particolare e infatti consente di creare, inviare e scambiare rappresentazioni digitali di tutte le forme di denaro: dollari, euro, bitcoin, praticamente qualsiasi cosa.
Il software di Stellar funziona su una rete aperta e decentralizzata e gestisce milioni di transazioni ogni giorno. Come Bitcoin ed Ethereum, Stellar si affida alla tecnologia blockchain per mantenere sincronizzata la rete. Stellar è molto più veloce, più economico e più efficiente dal punto di vista energetico rispetto ai tipici sistemi basati su blockchain. Stellar consente ad un utente di creare una rappresentazione riscattabile e negoziabile di qualsiasi risorsa (*asset*). Tali rappresentazioni sono chiamati *token*. Poiché qualsiasi account può emettere una risorsa sulla rete Stellar e qualsiasi utente può configurare un account, allora chiunque può emettere una risorsa: banche, società di servizi monetari di ogni tipo, imprese a scopo di lucro, organizzazioni non profit, comunità locali e persino individui. Una volta che una risorsa è stata creata si può cominciare a effettuare delle azioni con essa.
Le azioni su Stellar, come inviare pagamenti o fare offerte di acquisto o vendita, sono chiamate operazioni. Ogni operazione deve passare un controllo di validità, ovvero deve rispettare determinate condizioni. Per inviare un'operazione alla rete, questa deve essere raggruppata in una transazione. Quest'ultima è un gruppo da 1 a 100 operazioni accompagnate da alcune informazioni extra, come l'indicazione di quale account sta effettuando la transazione e una firma crittografica per verificare che la transazione sia autentica. Ogni transazione applicata al libro mastro comporta anche una piccola commissione (*fee*), necessaria per impedire ai malintenzionati di inviare spam alla rete.
Stellar è una raccolta di nodi *Stellar Core*, i quali sono computer che mantengono un registro comune di conti e saldi. Questi computer ascoltano le transazioni in entrata e, utilizzando lo *Stellar Consensus Protocol* (SCP), accettano di applicare una serie valida di tali transazioni per aggiornare il registro.

### Architettura distribuita

Ogni transazione viene sottoposta alle rete Stellar per essere registrata su un libro mastro proprio come avviene nella contabilità. Il ledger contiene una lista di tutti gli account. Per ogni titolare di account, il registro distribuito di Stellar memorizza due cose importanti:

- ciò che possiede (saldi del conto, come ad esempio "100 euro token " o "5000 lumen")
- cosa vuole fare con ciò che possiede (operazioni, come ad esempio "vendere 10 euro token per 50 lumen" o "inviare 100 token di qualsiasi tipo ad un altro conto").

Ogni 3-5 secondi, tutti i saldi e tutte le operazioni vengono trasmessi alla rete e immediatamente risolti. L'innovazione di Stellar è l'idea per cui non esiste un unico contabile ma vi è una rete di computer indipendenti, chiamati anche nodi, che controllano e ricontrollano il lavoro degli altri, il che significa che nessuno può fermare la rete o modificare segretamente i numeri a suo piacimento. Un algoritmo unico chiamato Stellar Consensus Protocol, che verrà spiegato nel dettaglio successivamente, mantiene il tutto sincronizzato.

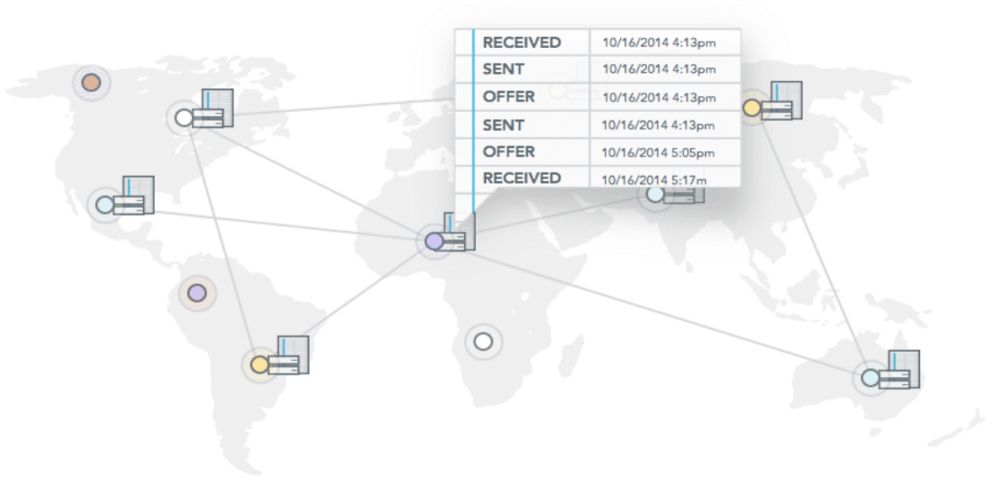


Figura 1.4: Rappresentazione distributed ledger

### Nodi

I nodi e il modo in cui comunicano sono informazioni pubbliche e chiunque può installare il software distribuito da Stellar e partecipare al processo di consenso. I nodi svolgono delle funzioni base, quali: eseguire Stellar Core, connettersi agli altri nodi, inviare transazioni, memorizzare lo stato del libro mastro in un database SQL e conservare una copia del libro mastro in file XDR. Oltre a queste funzioni base, esistono due opzioni di configurazione chiave che determinano il comportamento di un nodo: la partecipazione al consenso per la convalida delle transazioni e la pubblicazione di un archivio che altri nodi possono consultare per trovare lo storico completo della rete. Sulla base di queste due opzioni vengono definiti tre tipi di nodi:

- **Validatore base**
- **Validatore completo**
- **Archiviatore**

| TYPE OF NODE | SUPPORTS HORIZON | SUBMITS TRANSACTIONS | VALIDATES TRANSACTIONS | PUBLISHES HISTORY |
|---|---|---|---|---|
| Basic Validator | ✓ | ✓ | ✓ | |
| Full Validator | ✓ | ✓ | ✓ | ✓ |
| Archiver | ✓ | ✓ | | ✓ |

Figura 1.5: Rappresentazione delle tre tipologie di nodi Stellar

Un **validatore base** tiene traccia del libro mastro e sottomette le transazioni per una possibile integrazione, ma non è configurato per partecipare alla pubblicazione degli archivi storici. Richiede una chiave segreta ed è configurato per partecipare al consenso votando, e firmando, le modifiche al registro, ovvero supporta la rete e aumenta la decentralizzazione.

Un **validatore completo** esegue le stesse operazioni di un validatore base. La differenza con la configurazione precedente è quella per cui il validatore completo pubblica anche un archivio storico contenente istantanee del libro mastro, comprese tutte le transazioni ed i loro risultati. Un validatore completo scrive su AWS o Azure quindi è un po' più costoso e complesso da eseguire, ma fa anche il massimo per sostenere la resilienza e la decentralizzazione della rete. Quando altri nodi si uniscono alla rete possono consultare gli archivi offerti dai validatori di questo tipo per recuperare la storia della rete. Gli archivi ridondanti prevengono un *Single Point Of Failure* e permettono ai partecipanti della rete di verificare la veridicità di una data storia di transazioni.
Gli **archiviatori** aiutano la decentralizzazione offrendo conti ridondanti della storia della rete, ma non votano nè firmano i registri, quindi la loro utilità risulta abbastanza limitata.

### Horizon API

*Horizon* è il software responsabile che fornisce API HTTP ai dati nella rete Stellar. Raccoglie e conserva i dati prodotti dalla rete Stellar in una forma più facile da utilizzare per un'applicazione media rispetto alle rappresentazioni di dati orientate alle prestazioni usate da Stellar Core. La *Stellar Development Foundation* gestisce due server Horizon, uno per la rete pubblica e uno per la rete di test: "https://horizon.stellar.org" e "https://horizon-testnet.stellar.org". Questi server sono gratuiti per chiunque e dovrebbero andare bene per lo sviluppo di progetti su piccola scala. Sono, tuttavia, limitati e si raccomanda di non usarli per servizi di produzione che hanno bisogno di una forte affidabilità. Per questo motivo è preferibile l'esecuzione di Horizon all'interno della propria infrastruttura, il quale fornisce una serie di vantaggi tra cui:

- disabilita la limitazione del tasso di richiesta per un accesso di rete garantito,
- offre il pieno controllo operativo senza dipendere dalla Stellar Development Foundation,
- rende possibile l'esecuzione di istanze multiple per aumentare ridondanza e scalabilità.

Progetti come *wallet*, scambi decentralizzati ed emittenti di asset di norma usano Horizon per inviare transazioni, interrogare il saldo di un account o trasmettere eventi come le transazioni.

### Account

Gli account sono la struttura dati centrale in Stellar. Essi contengono i saldi, consentono di inviare e ricevere pagamenti, di inserire offerte di acquisto e vendita di beni, firmare transazioni ed emettere beni.
Stellar utilizza la crittografia a chiave pubblica per garantire che ogni transazione sia sicura: ogni account Stellar ha una coppia di chiavi composta da una chiave pubblica (una sequenza di 56 caratteri che inizia con la lettera "G") ed una chiave segreta (una sequenza di 56 caratteri che inizia con la lettera "S"). La chiave pubblica è visibile sul registro, chiunque può cercarla ed è ciò che gli altri account usano come indirizzo per inviare pagamenti sul conto, identificare l'emittente di un bene e verificare che una determinata transazione sia autorizzata. La chiave pubblica è sempre sicura da condividere. La chiave segreta dà la possibilità di accedere al conto, firmare transazioni, inviare fondi, ecc.. È come una password e non va mai condivisa. Una coppia di chiavi valida, tuttavia, non è sufficiente come requisito per la creazione di un account: per evitare che gli account inutilizzati gonfino il registro, Stellar richiede che gli account contengano un saldo minimo di 1 XLM.

### Operazioni

Le operazioni su Stellar rappresentano singoli comandi che mutano il registro. Le transazioni, che gli account firmano e inviano per l'inclusione nel ledger, sono in realtà gruppi di operazioni. Le transazioni possono, per definizione, includere da 1 a 100 operazioni. La capacità della rete, determinata dal voto del validatore, è misurata in termini di operazioni/registro. Attualmente, è impostata su 1.000 operazioni al secondo (equivalenti a 10 transazioni). Ogni operazione rientra in una specifica categoria di soglia: bassa, media o alta. Le soglie definiscono il livello di privilegio di cui ha bisogno un'operazione per avere successo.

### Validità di un'operazione

Quando una transazione viene inviata a un nodo, il nodo verifica la validità di ogni operazione nella transazione prima di tentare di includerla in un

set di transazioni candidate. In questa prima fase affinché un'operazione sia considerata valida, deve soddisfare le seguenti condizioni:

1. Le firme sulla transazione devono essere valide per l'operazione. Questo significa che le firme provengono da firmatari validi per l'account di origine dell'operazione ed inoltre che il peso combinato di tutte le firme per l'account di origine soddisfa la soglia per l'operazione.

2. L'operazione stessa deve essere ben formata. In genere questo significa controllare i parametri per l'operazione per vedere se sono in un formato valido (ad esempio, è possibile impostare solo valori positivi per l'importo di un'operazione di pagamento).

3. L'operazione deve essere valida nella versione del protocollo corrente della rete. Le operazioni deprecate infatti non sono valide.

**Transazioni**

Le transazioni sono comandi che modificano lo stato del libro mastro. Sono costituite da un elenco di operazioni compreso tra 1 e 100 e sono firmate, inviate alla rete e considerate per l'inclusione nel set di transazioni tramite SCP.
Ogni transazione ha i seguenti attributi:

- **Account di origine** : il conto che origina la transazione. Questo conto paga la commissione e fornisce il numero di sequenza per la transazione.

- **Fee** : ogni transazione comporta il pagamento di una commissione. Quando viene inviata una transazione, viene impostato l'importo massimo che si è disposti a pagare per operazione, ma viene addebitata poi la minima tariffa possibile in base all'attività di rete.

- **Numero di sequenza** : ogni transazione ha un numero di sequenza associato al conto di origine. Le transazioni seguono una regola rigida di ordinazione al fine di evitare il problema della doppia spesa. Quando si invia una singola transazione, è necessario inviare un numero di sequenza maggiore di 1 del numero di sequenza corrente. Ad esempio, se il numero di sequenza sul conto è 4, la transazione in entrata dovrebbe avere un numero di sequenza di 5. Tuttavia, se più transazioni con lo

stesso conto di origine entrano nello stesso set di transazioni, vengono ordinate e applicate in base al numero di sequenza. Ad esempio, se vengono inviate 3 transazioni che condividono lo stesso conto di origine e il conto è attualmente al numero di sequenza 5, le transazioni devono avere i numeri di sequenza 6, 7 e 8.

- **Elenco delle operazioni** : le transazioni contengono un elenco arbitrario di operazioni al loro interno. Tipicamente c'è una sola operazione, ma è possibile averne fino a 100. Le operazioni vengono eseguite in ordine e in maniera atomica, il che significa che o tutte le operazioni vengono applicate oppure nessuna lo è. Se un'operazione fallisce, l'intera transazione fallisce.

- **Elenco delle firme** : è possibile allegare fino a 20 firme per una singola transazione. Una transazione è considerata non valida se include firme non necessarie per autorizzare la transazione: le firme superflue non sono consentite. Le firme sono necessarie per autorizzare le operazioni e per autorizzare le modifiche all'account di origine.

- **Promemoria** : può contenere informazioni facoltative aggiuntive .

- **Limiti di tempo** : Il *timestamp* UNIX opzionale. Nel caso in cui una transazione viene inviata o troppo presto o troppo tardi, non verrà inserita nel set di transazioni. Se una transazione non viene inserita nel set di transazioni, viene conservata in memoria per essere aggiunta alla successiva serie di transazioni.

Una volta che una transazione è pronta per essere firmata, l'oggetto della transazione viene racchiuso in un *Transaction Envelope* che contiene la transazione e una serie di firme. Per determinare se una transazione è valida, vengono effettuati molti controlli agli attributi nel corso del ciclo di vita della transazione.

### Wallet

Un wallet Stellar è un'interfaccia, generalmente un'app, che consente ad un utente di accedere al proprio account, creare e firmare transazioni. Tale

accesso è controllato dalla chiave segreta dell'account. A differenza dei portafogli fisici in cui la risorsa, o il denaro, è archiviato all'interno del portafoglio, i portafogli di criptovaluta non memorizzano nè conservano gli assets. I wallet, invece, memorizzano la chiave privata, idealmente in modo sicuro, e/o firmano transazioni attraverso questa. Le risorse invece sono associate all'account nel libro mastro globale di Stellar. Un portafoglio solitamente offre un modo per visualizzare l'account, le risorse e i dati associati a tale account. Spesso aiuta anche a comporre le transazioni oltre che a firmarle e ad inviarle alla rete. Il portafoglio memorizza cache o riferimenti al database Stellar, ma i dati effettivi vengono archiviati sulla blockchain.

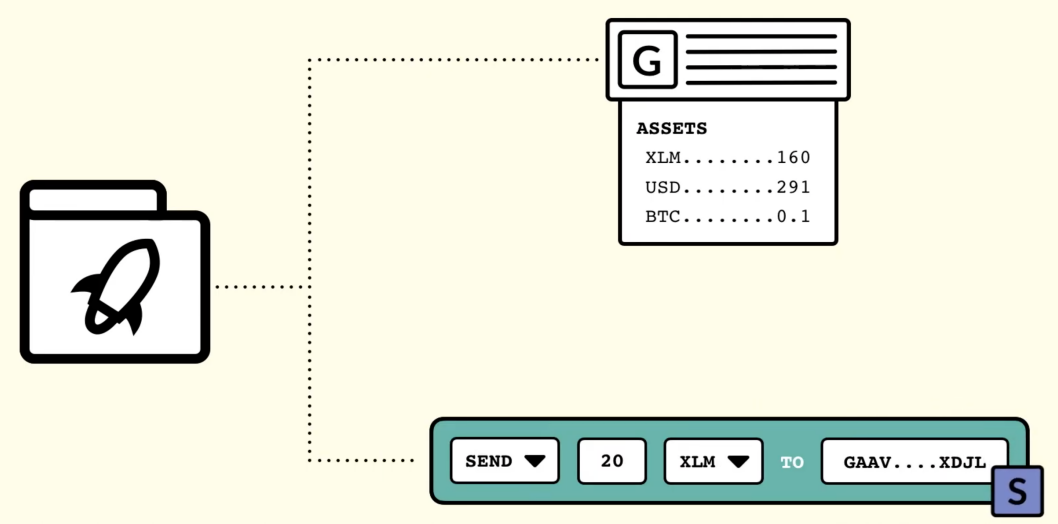


Figura 1.6: Rappresentazione di uno Stellar wallet

Tuttavia i portafogli operano lato client con le chiavi segrete dell'utente, dando accesso diretto agli account dell'utente. Pertanto, per la sicurezza del portafoglio, è necessario convogliare tutto il traffico web su metodi *Transport Layer Security* potenti. Poiché un portafoglio Stellar è un'interfaccia che consente ad un utente di accedere a un account memorizzato nel registro e poiché tale accesso è controllato solamente dalla chiave segreta dell'account, è molto importante la gestione della chiave privata.

**Asset's issuers**

Caratteristica fondamentale della rete Stellar è quella per cui ogni account ha la possibilità di emettere risorse. Qualsiasi asset può essere "tokenizzato" e trasferito sulla rete Stellar in modo rapido ed economico. Sono presenti tre tipi di asset nella rete:

- **Lumens**: sono la valuta nativa della rete e fungono da meccanismo di prevenzione dello spam tramite commissioni di transazione e come valuta ponte tra diverse risorse sull'exchange decentralizzato.
- **Non-Anchored Assets**: questi sono asset che hanno origine sulla rete e non rappresentano alcun asset al di fuori di essa (ad esempio token di utility).
- **Anchored Assets**: assets che non hanno avuto origine sulla rete ma sono collegati alla rete da un'entità di ancoraggio chiamata *Anchor*. Questi assets sono garantiti, rimborsabili o vincolati (ad esempio valuta Fiat).

Ogni asset in Stellar ha due caratteristiche: il codice identificativo dell' asset e la chiave pubblica dell'account emittente (*issuer*). La combinazione di questi permette di identificare in modo univoco ogni asset.
L'emissione di un asset è un processo che richiede quattro operazioni: la creazione di un account per l'emissione, la creazione di un account per la distribuzione, la creazione di una linea di fiducia (*trustline*) ed infine una operazione di pagamento. L'account di emissione è l'origine dell'asset e sarà sempre collegato all'identità dell'asset. L'account di distribuzione è il primo destinatario dell'asset. La risorsa verrà creata inviando un pagamento dall'account di emissione all'account di distribuzione.
I motivi per cui è necessario avere account separati per l'emissione e la distribuzione sono i seguenti:

- Sicurezza
- Verifica dei conti

Nel caso in cui l'account che viene usato per distribuire l'asset fosse anche l'account emittente e venisse compromesso da un attore malintenzionato, quell'attore potrebbe emettere assets senza limiti. Se invece l'account che viene

utilizzato per distribuire l'asset non è lo stesso di emissione, una volta scoperto l'attacco malevolo, l'account emittente potrebbe bloccare il saldo dell'account compromesso e ricominciare da capo con un nuovo account di distribuzione. Ciò sarebbe possibile senza il bisogno di modificare l'account issuer e garantirebbe una maggior sicurezza del sistema. La seconda ragione è la tenuta della contabilità o la verificabilità di questa. L'account di emissione non può effettivamente tenere un saldo del proprio asset. Se si possedesse un bilancio permanente dell'asset in un account separato, allora sarà più facile da monitorare lo stato della risorsa emessa.
Prima che un account possa detenere un asset emesso da un altro account,come anticipato in precedenza, deve stabilire una trustline.

## 1.4 Consenso

Raggiungere un consenso è assai difficile nella quotidianità, altrettanto lo è all'interno di un sistema informatico. In quest'ultimo, infatti, il problema sorge in particolare quando diversi componenti hanno visioni potenzialmente divergenti sullo stato del sistema e su ciò che sta accadendo al suo interno. Un dato comportamento coerente complessivamente può essere raggiunto solo quando tutti i componenti sono in accordo su ciò che è rilevante per il sistema stesso, come ad esempio eventi.
Il consenso, all'interno di questo elaborato di tesi, è il processo attraverso il quale viene raggiunto un accordo sullo stato del sistema tra macchine inaffidabili collegate da reti asincrone.
L'idea di una rete basata sul consenso nasce dall'esigenza di garantire equità e stabilità all'interno del sistema anche in presenza di nodi difettosi e inaffidabili. Per raggiungere tale scopo, è necessario che i nodi siano in grado di coordinarsi al fine di accordarsi su una o più affermazioni.
In una rete che gestisce anche criptovalute, come ad esempio quella di Stellar, i nodi devono ripetutamente decidere quale sia la storia completa del Distributed Ledger, tra più versioni possibili che occasionalmente sono in conflitto. Questo accordo permette a chi riceve una criptovaluta di avere fiducia che questa sia valida e non sia già stata spesa altrove. Inoltre assicura che sarà in grado di spenderla in futuro, perché il nuovo destinatario avrà la stessa fiducia in essa, per le stesse ragioni.
Un protocollo di consenso fornisce la proprietà di *safety* se tutti gli output prodotti hanno lo stesso valore e questo valore di output sarà poi quello corretto. Un protocollo di consenso espone la proprietà di *liveness* se alla fine i nodi che non falliscono producono un valore. La proprietà di *fault-tolerance* è presente se un certo protocollo può recuperare dal fallimento di un nodo in qualsiasi momento. Il teorema di risultato di impossibilità FLP indica che: "Nessun protocollo di consenso deterministico può garantire tutte le proprietà di safety, liveness e fault-tolerance in un sistema asincrono".
Per garantire la proprietà di safety è sufficiente che ogni due quorum ci sia un nodo qualsiasi in comune.
Qui di seguito viene contestualizzato il consenso all'interno di due sistemi, il

*Byzantine Agreement System* e il *Federated Byzantine Agreement System.*

### BAS - Byzantine Agreement System

Un sistema di accordo di tipo bizantino è caratterizzato dall'essere tollerante ai cosiddetti "errori bizantini": si tratta di nodi che danno informazioni false, e questo può accadere sia per errore sia nel tentativo deliberato di sovvertire il sistema per ottenere qualche vantaggio.
Cosa succede se si ha un *Byzantine Failure*?
Nel Byzantine Agreement System, per garantire la proprietà di safety, non è più sufficiente che ogni due quorum ci sia un nodo qualsiasi in comune, bensì è necessario che sussista almeno un nodo *non-faulty* in comune.
In questo sistema è noto a priori il numero di nodi presenti. Ipotizzando un sistema con **N** nodi e con la dimensione del quorum uguale a **T**, la sovrapposizione minima tra due quorum sarà uguale a **2T-N**. Per soddisfare la proprietà di safety e garantire tolleranza ai guasti, il numero massimo di fallimenti bizantini che il sistema è in grado di tollerare sarà **2T-N-1**.
Per soddisfare la proprietà di liveness, e quindi continuare a fare progressi, è necessario avere un quorum. Considerando, come citato in precedenza, la dimensione del quorum uguale a **T**, allora è possibile tollerare fino a **N-T** fallimenti bizantini e avere ancora un quorum residuo valido.
Tipicamente, ciò che si fa è impostare il numero di nodi, $\mathbf{N = 3f+1}$, e la dimensione del quorum, $\mathbf{T = 2f+1}$, per qualche intero **f**. Questo consente di ottenere il punto di equilibrio **f** in cui il numero di fallimenti che garantiscono safety è uguale al numero di fallimenti che garantiscono liveness.

### FBAS - Federated Byzantine Agreement System

Un sistema di accordo federato bizantino è una specializzazione del BAS, caratterizzato dall'assenza di un accordo universale riguardo chi siano i nodi e su quanto essi siano affidabili. L'idea chiave in un FBAS è quindi quella di stabilire i quorum in maniera decentralizzata e distribuita.
Qualsiasi sistema di accordo in una rete informatica distribuita deve essere tollerante agli errori. Deve quindi produrre risultati coerenti nonostante errori come collegamenti di comunicazione lenti, nodi che non rispondono e messaggi

disordinati.
Il protocollo descritto nella sezione successiva, lo Stellar Consensus Protocol, è un Federated Byzantine Agreement System che consente alle reti informatiche decentralizzate e senza leader di raggiungere in modo efficiente un risultato di consenso su alcune decisioni. Un nodo in una rete di questo tipo si rifiuta di impegnarsi in una particolare versione della cronologia finché non vede che anche un numero sufficiente di suoi pari, un quorum, è pronto a farlo.
Una volta che accade ciò, i nodi hanno formato un insieme di voti abbastanza grande da costringere i restanti nodi della rete a concordare con la loro decisione. Un nodo bizantino potrebbe essere in grado di far mentire alcuni nodi per suo conto, ma se la rete è abbastanza grande, il suo tentativo sarà sopraffatto dai voti dei nodi onesti.
Quindi quanti nodi servono per formare un quorum? Per combattere errori e frodi è richiesta almeno una maggioranza semplice, e più tipicamente una maggioranza qualificata . Capire quando si ha la maggioranza significa sapere a priori quanti partecipanti fanno parte della rete. Tuttavia, se l'insieme di partecipanti è una rete FBAS, cioè vagamente definita e in cui i membri possono unirsi e lasciare la rete senza bisogno di coordinarsi con alcuna autorità centrale, allora è richiesto un sistema di accordi bizantini federati: ovvero un sistema che possa determinare i quorum non tramite un elenco predeterminato di nodi , bensì in maniera dinamica.
Potrebbe non sembrare possibile costituire un quorum dalla sola prospettiva limitata di un singolo nodo in una rete tentacolare, ma lo è.
Il whitepaper SCP mostra come farlo utilizzando una procedura chiamata votazione federata. Per poter comprendere meglio il funzionamento di SCP è prima necessario spiegare la procedura di voto federato descritta anche all'interno del whitepaper di David Mazières.

### Votazione federata

Il voto federato è una procedura per valutare se una rete di partecipanti riesce a raggiungere la medesima decisione su un dato oggetto. Nella votazione federata, ogni nodo deve scegliere uno dei potenziali valori possibili come risultato di quel round. Esso non può farlo finché non è sicuro che gli altri nodi della rete sceglieranno il medesimo risultato. Per avere tale certezza, i nodi

si scambiano una serie di messaggi reciproci consentendo a ciascuno di loro di confermare che un quorum di nodi accetta lo stesso voto.
Come discusso in precedenza, in una rete decentralizzata con partecipazione dinamica, è impossibile sapere in anticipo quanti nodi ci sono e quindi quanti di questi costituiscono la maggioranza. Il voto federato risolve questo problema introducendo la nuova idea di "porzione di quorum": cioè una piccola raccolta di peer di cui un nodo si fida per trasmettere informazioni sullo stato del voto nel resto della rete. Ogni nodo definisce la propria fetta di quorum. Per poter formare il quorum generale del sistema ogni nodo aggiunge i peer della propria porzione di quorum. Quindi si aggiungono i peer delle porzioni di quei nodi e così via. Proseguendo iterativamente in questo modo, si avranno sempre più nodi che non si possono aggiungere perché sono già stati inclusi nel quorum. Quando non ci sono più nuovi nodi da aggiungere bisogna fermarsi. In questo modo si è formato il quorum grazie ad una "chiusura transitiva" della sezione di quorum del nodo di partenza.
Ogni peer può possedere più di una quorum slice. Per formare il quorum, si sceglie solo una delle slice. Questo significa che ogni nodo fa parte di molti quorum possibili.
È importante ricordare che in un sistema di accordo bizantino non federato, un quorum è definito come la maggioranza di tutti i nodi. Una volta che una proposta supera la soglia del quorum, il resto dei membri della rete è convinto che qualsiasi proposta in disaccordo fallirà. È in questo modo che la rete converge su un unico risultato.
In un sistema di accordi bizantino federato però, non solo non può esserci maggioranza (perché nessuno conosce la dimensione totale della rete), ma il concetto di maggioranza non è nemmeno utile. Se l'adesione al sistema è aperta, qualcuno potrebbe ottenere la maggioranza semplicemente conducendo un cosiddetto attacco *Sybil*: collegandosi alla rete molte volte utilizzando più nodi. Quindi cosa c'è nella chiusura transitiva della sezione di quorum di un nodo che lo trasforma in un quorum e cosa lo rende in grado di sopraffare le proposte concorrenti? Tecnicamente, niente! Immaginando una rete contenente Alice, Bob, Carol, Dave, Elsie e Frank. Alice ha Bob e Carol nella sua fetta di quorum. Bob ha Alice e Carol, Carol ha Alice e Bob. Nel frattempo, Dave, Elsie e Frank hanno tutti l'un l'altro nelle rispettive quorum slice. Il

sottogruppo Alice-Bob-Carol può raggiungere una decisione di cui il gruppo Dave-Elsie-Frank non sentirà mai parlare e viceversa. Non c'è modo per questa rete di ottenere consenso, se non per puro caso.
Quindi SCP richiede, affinché il voto federato funzioni, che la rete goda di una proprietà chiamata intersezione del quorum. In una rete con questa proprietà, è possibile costruire sempre due quorum che si sovrappongono in almeno un nodo.
Intuitivamente, significa che se un qualsiasi quorum è d'accordo con l'affermazione **X**, nessun altro quorum potrà mai essere d'accordo con **non-X**, perché includerà necessariamente qualche nodo del primo quorum che ha già votato per **X**.
Naturalmente potrebbe accadere che i nodi facenti parte dell'intersezione siano tutti bizantini. In tal caso, avere l'intersezione del quorum non aiuta affatto la rete ad essere d'accordo. Per questo motivo, il whitepaper di Stellar si basa su assunzioni esplicitamente dichiarate, come ad esempio che la rete goda dell'intersezione del quorum anche se i nodi che si comportano in modo anomalo vengano rimossi dalla rete.
Per motivi di chiarezza, saranno lasciate tali assunzioni implicite per il resto di questa tesi.
Potrebbe sembrare irragionevole aspettarsi che una raccolta di nodi indipendenti organizzi le proprie sezioni in modo tale che la rete possa godere in modo affidabile dell'intersezione del quorum. Ci sono però due ragioni per cui questo non è così inverosimile.
La prima ragione è l'esistenza di Internet stessa. Internet è l'esempio perfetto di una rete di nodi indipendenti con intersezione del quorum. La maggior parte dei nodi su Internet si connette solo a pochi altri nodi locali, ma quei piccoli insiemi si sovrappongono abbastanza da rendere ogni nodo raggiungibile da tutti gli altri.
Il secondo motivo è specifico della rete di pagamento Stellar. Qualsiasi tipo di asset nella rete Stellar ha un issuer e le best-practice di Stellar richiedono che ogni issuer designi uno o più nodi nella rete per la gestione delle richieste. È interesse dell'issuer includere quei nodi nelle sezioni del suo quorum, direttamente o indirettamente, per ogni risorsa a cui tiene. I quorum per tutti i nodi interessati alla stessa risorsa si sovrapporranno quindi almeno in quei

nodi. I nodi interessati a più asset includeranno nelle loro sezioni di quorum tutti i nodi di gestione degli issuer rilevanti e questi tenderanno a collegare tutti gli asset insieme. Non è necessario, inoltre, che tutte le risorse che non sono connesse in questo modo ad altre sulla rete lo siano: è normale che la rete non abbia l'intersezione del quorum in quel punto. Naturalmente, aspettarsi che la rete goda dell'intersezione del quorum non equivale ad una garanzia.
In un round di votazione federata, un nodo inizia facoltativamente votando per un valore **V**. Questo voto significa trasmettere un messaggio alla rete che dice: "*Sono il nodo N, le mie quorum slice sono Q e io voto V*". Quando un nodo vota in questo modo, promette di non aver mai votato contro **V** e di non farlo nemmeno in futuro.
Un nodo può vedere come i suoi peer votano tramite i loro messaggi broadcast. Una volta che questo nodo ha raccolto un numero sufficiente di messaggi di questo tipo, può vedere le sezioni del quorum al loro interno per trovare i quorum. Se può vedere un quorum di peer che votano tutti anche per **V**, allora può passare ad accettare **V** e trasmettere questo nuovo messaggio alla rete: "*Sono il nodo N, le mie sezioni di quorum sono Q e accetto V.*"
L'accettazione fornisce una garanzia più forte del semplice voto. Quando un nodo vota per **V**, non potrà mai votare per **non-V**. Ma quando un nodo accetta **V**, nessun nodo della rete accetterà mai **non-V**.
Naturalmente, ci sono buone probabilità che il nodo **N** non veda subito un quorum di nodi che concorda con il suo voto **V**. Altri nodi infatti possono votare per valori diversi.
C'è un altro modo per **N** di passare dal semplice voto all'accettazione. Il nodo può accettare un valore diverso, **W**, anche se lui stesso non lo ha votato, e anche se non vede un quorum che lo vota, purché veda un set di blocco che lo accetta. Un set di blocco è composto da un nodo scelto da ciascuna delle sezioni del quorum di **N**. Se tutti i nodi in un tale insieme accettano **W**, allora non sarà mai possibile formare un quorum che accetti **non-W**, e quindi è sicuro che anche **N** accetti **W**.
Ma un set di blocco non è un quorum. Sarebbe troppo facile per qualcuno ingannare il nodo **N** nell'accettare un valore quando non dovrebbe, potendo semplicemente sovvertire un nodo in ciascuna delle sezioni di **N**. Per questo l'accettazione di un valore non rappresenta la fine della votazione federata.

Infatti, il nodo **N** deve poi confermare il valore, il che significa che deve vedere un quorum di nodi che accettano tutti lo stesso valore.
Se riesce a far ciò, come dimostra il whitepaper SCP, allora anche il resto della rete confermerà lo stesso valore, e quindi **N** ha raggiunto la fine del voto federato con il valore confermato come risultato.
Il processo di voto, accettazione e conferma costituisce un round completo di voto federato. Lo Stellar Consensus Protocol combina molti di questi round per creare un sistema di consenso completo.

### 1.4.1 Stellar Consensus Protocol

La procedura di voto federato appena descritta è sicura nel senso che se un nodo conferma il valore **V**, nessun altro nodo confermerà un valore diverso. Ma "non confermerà un valore diverso" non è lo stesso concetto di "confermerà qualcosa". Potrebbero esserci così tanti valori diversi da votare che nessuno di questi raggiunge la soglia di accettazione. Ciò significa che la votazione federata manca di liveness. Lo Stellar Consensus Protocol utilizza il voto federato in modo che riesca a garantire safety e liveness. L'idea descritta nel whitepaper è quella di condurre più votazioni federate su più valori fino a quando uno di questi non supera le varie fasi di voto di SCP, descritte qui di seguito.
I valori su cui SCP cerca consenso possono essere registri della rete Stellar o qualsiasi altra cosa, ma è importante notare che questi non sono i valori su cui votano, accettano o confermano i turni di voto federati di SCP. Invece, il voto federato avviene su dichiarazioni su quei valori.
I primi turni di votazione federata avvengono nella fase di nomina, su una famiglia di dichiarazioni della forma, "*Nomino V*", per possibilmente molti diversi valori di **V**. L'obiettivo della nomina è trovare una o più di tali dichiarazioni che possono rendere tutto il percorso attraverso l'accettazione e la conferma.
Dopo aver trovato nominativi confermabili, SCP passa alla fase di ballottaggio, dove l'obiettivo è trovare qualche scheda elettorale (che è un contenitore per un valore nominato) e un quorum che possa impegnarsi ad essa. Se un quorum si impegna a questo scrutinio, il suo valore è il risultato di questo round di consenso. Ma prima che un nodo possa votare per impegnarsi in una votazione, deve prima confermare che tutte le votazioni minori sono state

annullate. Questi passaggi comportano più turni di votazione federata su più dichiarazioni sulle schede elettorali. Le sezioni seguenti descrivono la nomina e il ballottaggio in maniera più dettagliata.

### Nomina

All'inizio della fase di nomina, ogni nodo può scegliere spontaneamente un valore **V** e votare per l'affermazione "*Nomino V*".
L'obiettivo in questa fase è confermare la nomina di un qualche valore, tramite il voto federato.
È possibile che un numero sufficiente di nodi voti per un numero sufficiente di affermazioni "*Io nomino*" diverse: nessuna nomina può mai raggiungere la soglia di accettazione. Quindi, oltre a esprimere i propri voti di nomina, i nodi fanno eco delle nomine dei loro pari. Fare l'eco significa che un nodo che vota "*Nomino V*", dopo aver visto un messaggio da un peer che nomina **W**, ora vota sia "*Nomino V*" che "*Nomino W*".
Concettualmente si tratta di voti federati differenti che si verificano in parallelo, ciascuno separatamente in grado di raggiungere l'accettazione o la conferma. In pratica, i messaggi del protocollo raggruppano insieme questi voti separati. Sebbene un voto per nominare **V** sia una promessa di non votare mai contro la nomina di **V**, il livello di applicazione al di sopra del voto federato, SCP in questo caso, arriva a definire cosa significa "contro". Il protocollo di consenso di Stellar non definisce alcuna affermazione che contraddica un voto "*Nomino X*", non esiste il voto "*Rifiuto la nomina di X*", quindi un nodo può votare per nominare tutti i valori che vuole. Molte di queste nomine potrebbero non giungere mai alla fase successiva, ma alla fine ce ne saranno una o più che un nodo può accettare o confermare. Una volta che un nominato è stato confermato, viene chiamato candidato.
La nomina può produrre più candidati confermabili, quindi SCP richiede che il livello di applicazione fornisca un metodo per combinare i candidati in un singolo risultato composto. Il metodo di combinazione può essere qualsiasi cosa richiesta dal livello dell'applicazione, purché tale metodo sia deterministico, il che significa che ogni nodo che combina gli stessi due candidati produce lo stesso risultato. Nella rete di pagamento Stellar, dove gli intestatari sono libri mastri, la combinazione di due intestatari proposti implica l'unione delle

transazioni che contengono e l'ultimo dei due timestamp.
Il whitepaper SCP dimostra, grazie al Teorema 12, che la nomina fa sì che la rete alla fine converga su un singolo candidato composto al termine della nomina. Ma c'è un problema: il voto federato è un protocollo asincrono.
In altre parole, i nodi non sono coordinati dal tempo, ma solo dai messaggi che inviano. Dal punto di vista di un nodo non è chiaro quando la nomina sia terminata. E sebbene tutti i nodi alla fine arrivino allo stesso candidato, possono prendere strade diverse per arrivarci, producendo candidati composti diversi e validi lungo la strada, senza mai essere in grado di dire quale sia quello finale.
Questa situazione però è accettabile perchè la nomina riguarda solo la produzione e la limitazione del numero di candidati per il consenso. Il resto del procedimento è gestito da un processo chiamato ballottaggio.

### Ballottaggio

Viene definita scheda elettorale una coppia: *<contatore,valore>*, dove il contatore è un numero intero che inizia da 1 ed il valore è un candidato della fase di nomina. Potrebbe essere il candidato del nodo stesso, o il candidato di qualche peer che il nodo arriva ad accettare. Il ballottaggio fa ripetuti tentativi per ottenere il consenso della rete su qualche candidato in qualche ballottaggio conducendo potenzialmente molte votazioni federate sulle dichiarazioni dei candidati. Il contatore tiene traccia dei tentativi effettuati e le schede con contatore più alto hanno la precedenza sulle schede con contatore più basso. Se la coppia *<contatore,valore>* sembra bloccarsi, inizia una nuova votazione per la coppia *<contatore+1,valore>*.
È importante distinguere tra valori, schede (coppia di contatore-valore) e dichiarazioni sulle schede. Un round di SCP include più fasi di votazione federata su tali dichiarazioni, in particolare queste:

- *"Sono 'preparato' a impegnarmi per la scheda B"*
- *"Mi 'impegno' per la scheda B."*

Dal punto di vista di un qualsiasi nodo, il consenso si raggiunge quando trova una scheda **B** per la quale è in grado di confermare (trovare un quorum che

accetta) l'affermazione *"Mi impegno alla scheda B"*.
A questo punto è sicuro agire sul valore in **B**. Questo fase viene chiamata esternalizzazione del valore. Una volta confermato l'impegno di una votazione, un nodo può essere sicuro che ogni altro nodo ha esternalizzato, o inevitabilmente esternalizzerà, lo stesso valore.
Sebbene concettualmente abbiano luogo numerose votazioni federate su dichiarazioni su molte schede diverse, in realtà vengono scambiati molti meno messaggi di protocollo effettivi, poiché ognuno incapsula una serie di schede. Un singolo messaggio quindi fa avanzare lo stato di molti voti federati contemporaneamente, ad esempio: *"Accetto che vengano impegnate le schede nell'intervallo da <min,V>a <max,V>"*. Quindi cosa significa dichiarare di "prepararsi" e "impegnarsi"? Un nodo vota per impegnarsi in una votazione quando è convinto che altri nodi non si impegneranno in votazioni con valori diversi. Convincersi di questo è lo scopo della dichiarazione di preparazione. Un voto che dice *"Sono preparato a impegnarmi per la scheda B"* è una promessa di non impegnarsi mai per una scheda inferiore a **B**, cioè uno con un contatore più piccolo. Si dice che quei voti minori siano eliminati dal voto di preparazione , mentre la scheda **B** è preparata.
SCP definisce "eliminazione" come l'opposto di "impegno". Un voto per preparare una scheda è implicitamente anche un voto per eliminare altre schede e, come abbiamo discusso in precedenza, votare per qualcosa è una promessa di non votare mai contro.
Prima di impegnarsi su una scheda, un nodo deve prima trovarne una che può confermare che è stata preparata. In altre parole, conduce il voto federato su "*Sono pronto a impegnarmi per la scheda B*" per potenzialmente molte schede diverse finché non ne trova uno che il quorum accetti.
Da dove vengono le schede per i voti di preparazione? Il primo voto di preparazione che un nodo emette è per *<1,C>*, dove **C** è il candidato composto prodotto dalla fase di nomina. Tuttavia, anche dopo l'inizio delle votazioni di preparazione, la nomina può produrre ulteriori candidati, quindi questi diventano nuovi voti. Nel frattempo, i peer potrebbero avere candidati diversi e potrebbero formare un set di blocco che accetta *"Sono pronto a impegnarmi per la scheda B2"*, che convincerebbe anche il nodo ad accettarlo.
Infine, c'è un meccanismo di timeout che fa iniziare nuovi turni di votazione

federata su nuove schede con contatori più alti se le schede attuali sembrano bloccate.
Una volta che un nodo trova una scheda **B** che può confermare che è preparata, esprime un nuovo voto per *"Mi impegno alla scheda B"*. Questo voto dice ai peer che il nodo non eliminerà mai **B**. Infatti, se **B** è il voto $<N,C>$ , allora *"Mi impegno al voto* $<N,C>$*"* è implicitamente anche un voto per preparare ogni scheda da $<N\ ,C>$ fino a $<\infty,C>$. Questo ulteriore significato aiuta i nodi che ricevono questo messaggio a recuperare il ritardo, se sono ancora nelle fasi precedenti del protocollo.
Vale la pena sottolineare nuovamente a questo punto che si tratta di protocolli asincroni. Solo perché un nodo sta dicendo di impegnarsi su una scheda non significa che siano nella stessa fase anche i suoi pari. Alcuni potrebbero ancora votare per preparare le dichiarazioni, altri potrebbero aver già esternalizzato un valore. SCP descrive quindi, come mostrato in figura 1.7, come un nodo dovrebbe gestire ogni tipo di messaggio ricevuto dai suoi peer durante il ballottaggio, indipendentemente dalla fase in cui si trova.

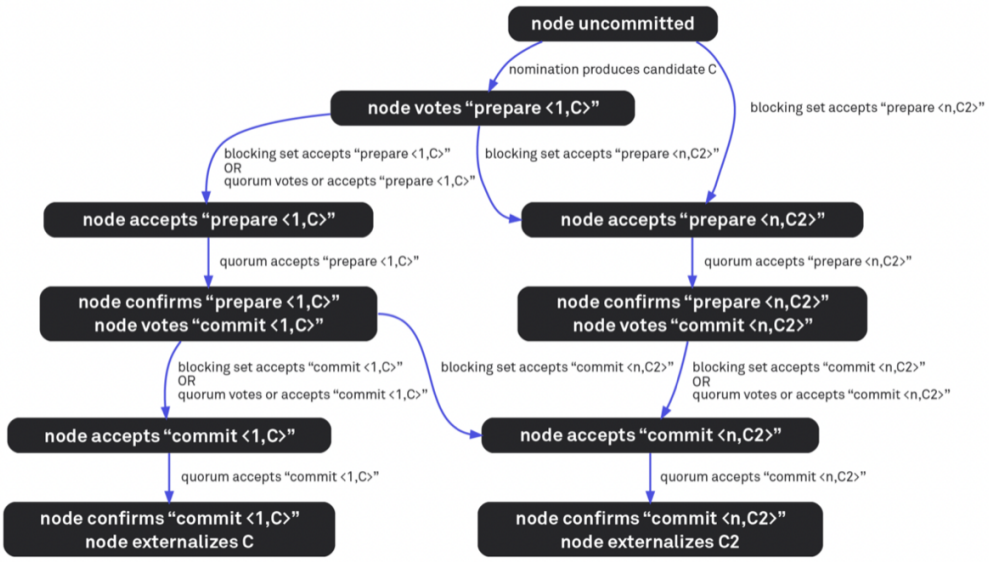


Figura 1.7: Rappresentazione del ballottaggio SCP

Se la dichiarazione *"Mi impegno a* $<N,C>$*"* non può essere accettata o confermata, forse $<N+1,C>$ può, oppure $<N+2,C>$, o comunque qualche scheda

con **C** dentro e nessun altro valore, poiché il nodo ora ha promesso di non eliminare mai <*N,C*>. Nel momento in cui un nodo esprime il proprio impegno per un valore, per quanto riguarda il consenso o viene accettato e confermato o si interrompe.
Tuttavia, questo non è ancora sufficiente per consentire al nodo di esternalizzare **C**. Alcuni peer bizantini, che ammontano a meno di un quorum in base ai nostri presupposti di sicurezza, potrebbero mentire al nodo. Accettare e poi confermare una scheda (o una serie di schede) è ciò che dà al nodo la fiducia di poter finalmente esternare il valore.
Una volta che la rete ha raggiunto il consenso, è pronta per ricominciare tutto da capo. Nella rete di pagamento Stellar, questo avviene all'incirca una volta ogni 5 secondi.
Riassumendo, SCP è in grado di fare tutto questo basandosi su più turni di voto federato. Il voto federato è reso possibile dal concetto di sezioni di quorum: insiemi di peer di cui ciascun nodo ha scelto di fidarsi come parte del proprio quorum. Il procedimento appena descritto garantisce quindi che è possibile arrivare al consenso, anche in una rete con appartenenza aperta e fallimenti di tipo bizantino.

### 1.4.2 Considerazion sul quorum

La ridondanza all'interno di un'organizzazione sulla rete Stellar nella configurazione del quorum è una buona pratica, ed è per questo che è raccomandata l'esecuzione di almeno tre validatori per garantire disponibilità. In precedenza, Stellar Core non distingueva tra validatori e organizzazioni, rendendo il singolo utente responsabile della corretta configurazione di slice e soglie.
Considerando un semplice esempio di configurazione delle quorum slice:
**- {A, B, C, D, E, F, G} - soglia 67%**
La soluzione potrebbe sembrare ragionevole. Sono presenti 7 nodi, e 5 di questi sono necessari per raggiungere il consenso. Tuttavia, cosa succede se oltre ai nomi dei nodi, vengono definiti anche i nomi dell'organizzazioni che li gestiscono?
**- {SDF1, SDF2, SDF3, SDF4, SDF5, LOBSTR, SATOSHIPAY} - soglia 67%**
Il problema ora diventa più visibile. Se LOBSTR e SATOSHIPAY non sono

d'accordo con ciò che dice SDF, quest'ultimo procederà comunque, poiché i suoi stessi nodi sono d'accordo e sono in numero sufficiente per soddisfare la soglia. Il problema è che le organizzazioni non sono ponderate correttamente perché ad alcune di esse capita di gestire più nodi di altre. Dal punto di vista del consenso, bisogna preoccuparsi di un'organizzazione nel suo complesso, non dei suoi singoli validatori.

Per riflettere questa preoccupazione, Stellar Core ora permette di specificare l'*HOMEDOMAIN* per ogni validatore. I nodi gestiti da un'organizzazione hanno lo stesso dominio, e sono automaticamente raggruppati insieme con una soglia di maggioranza semplice (51%). Se il dominio è specificato correttamente per ogni nodo, la configurazione di cui sopra diventa automaticamente:

**- {{SDF1, SDF2, SDF3, SDF4, SDF5 — threshold 51%}, LOBSTR, SATOSHIPAY} — threshold 67%**

In aggiunta al concetto di dominio, i validatori ora possono perfino specificare la qualità dei nodi nelle loro sezioni di quorum. La qualità è in qualche modo soggettiva, è una misura di quanto affidabile e degna di fiducia sia un'organizzazione.

Ci sono tre configurazioni possibili di qualità: *HIGH*, *MEDIUM* e *LOW*.

Un validatore di alta qualità è affidabile, pubblica un archivio storico e ha sempre avuto un buon comportamento sulla rete.

Un validatore di media qualità potrebbe essere meno affidabile o meno conosciuto. È un candidato per diventare un validatore di alta qualità se rimane affidabile, inizia a pubblicare un archivio e si guadagna il rispetto della rete centrale.

Un validatore di bassa qualità non è ancora conosciuto o considerato affidabile dalla rete. Se si dimostra affidabile e degno di fiducia nel tempo, può essere promosso ad una qualità superiore. Il seguente diagramma in figura 1.8 mostra come i nodi sono raggruppati dopo che è stata specificata una valutazione di qualità per ogni nodo nella sezione del quorum:

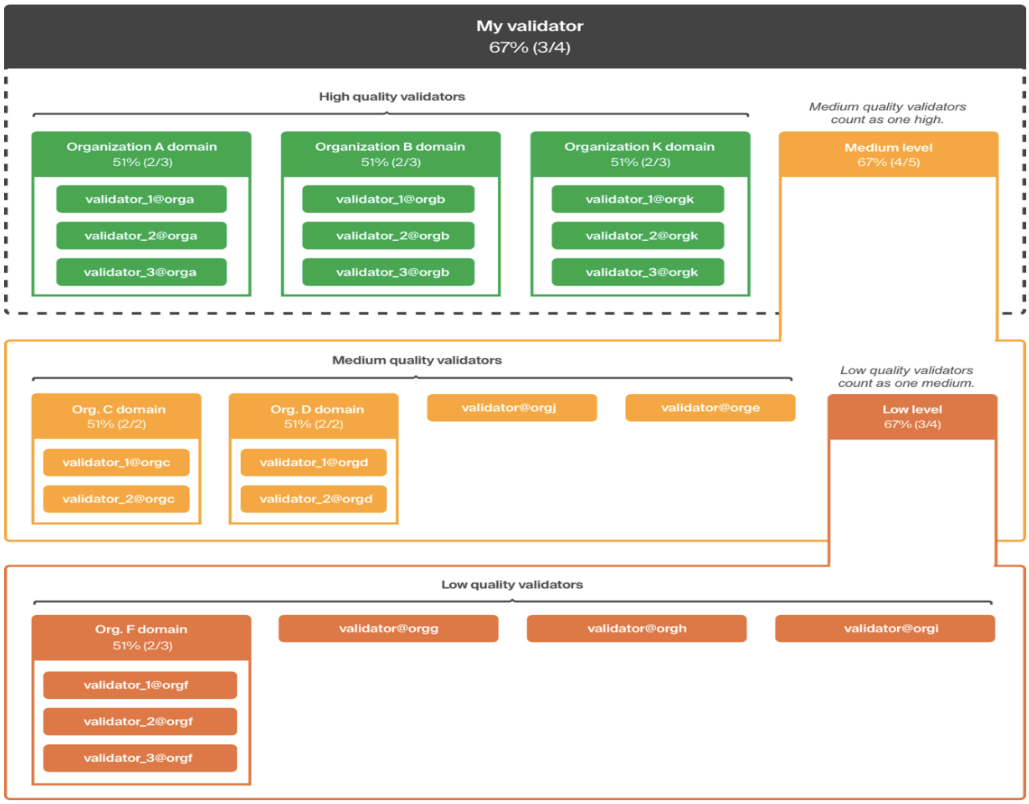


Figura 1.8: Rappresentazione della configurazione della qualità dei nodi

Strutturando i nodi in questo modo si incrementa la proprietà di liveness, poiché è più probabile che i nodi di alta qualità siano disponibili (a causa dei requisiti di affidabilità più severi). Nel caso in cui un nodo di alta qualità sia di tipo bizantino o smetta di funzionare, il gruppo di nodi di media qualità agisce come back-up per assicurare il consenso.
In pratica, l'accordo tra tutti i validatori di media qualità è uguale ad un voto nel livello di alta qualità. Allo stesso modo, l'accordo tra tutti i validatori di bassa qualità è uguale ad un voto nel livello di media qualità.

# Capitolo 2

# Analisi dei requisiti

## 2.1 Obiettivi del progetto

Senza rendercene conto, nella vita di tutti i giorni utilizziamo piattaforme, app e sistemi di pagamento *fintech*, e non ne possiamo più fare a meno. Oggi il fintech fa parte della nostra quotidianità. Ma cos'è esattamente il Fintech? Fintech è la fusione di finance e technology e include tutte quelle tecnologie che si pongono l'obiettivo di semplificare, migliorare e digitalizzare i servizi finanziari tradizionali.
Questa tesi descrive la progettazione e realizzazione di una rete blockchain permissioned capace di gestire transazioni di token applicabile a innumerevoli contesti all'interno di un ecosistema di pagamenti e di servizi. La tecnologia blockchain offre molteplici vantaggi tra cui la possibilità di diminuire le commissioni delle transazioni rispetto agli attuali sistemi di pagamento.
A seguito di un'analisi differenziale è stata scelta la tecnologia Stellar per la realizzazione della rete blockchain, in quanto presenta le caratteristiche più idonee a soddisfare le specifiche tecniche richieste. Stellar è una rete aperta e decentralizzata basata su blockchain per gestire transazioni di token, che consente il *fork*, ovvero la creazione di una rete "permissioned" partendo dalla genesi di nodi completamente indipendenti. Utilizza lo Stellar Consensus Protocol per garantire il livello di sicurezza e la validità delle transazioni emesse. Inoltre, rispetto alle blockchain più famose al momento (Bitcoin, Ethereum, Algorand) risulta più veloce ed economica, sia per quanto riguarda il costo

per eseguire le transazioni, sia per il consumo di energia e risorse hardware richieste.
In aggiunta a tutte le caratteristiche e le garanzie di sicurezza fornite implicitamente dall'utilizzo di questa tecnologia si è optato per rendere la rete privata. La blockchain permissioned realizzata è più vicina alle esigenze degli utilizzatori essendo più controllata ed anche più performante e veloce dell'equivalente blockchain pubblica. Offre inoltre un ecosistema di transazioni sorvegliato e con accesso limitato ai soggetti autorizzati. L'infrastruttura è infatti costituita da nodi appartenenti solamente ai partecipanti autorizzati a prendere parte al processo di consenso della rete, chiamati anche attori.
L'architettura dell'infrastruttura di rete progettata prevede, oltre ai nodi della rete blockchain vera e propria, altri server che si occupano in particolare di offrire un servizio di database di custodia delle chiavi, un servizio di load balancing ed un servizio di accesso ai dati presenti nella rete tramite chiamate API fornite dai nodi Horizon.
Oltre a questi è stato creato un elemento ad-hoc, ovvero il software BTKL, utilizzato per semplificare la comunicazione con la blockchain e per incrementare la sicurezza di comunicazione con il database di custodia.
Al fine di garantire l'assoluta sicurezza della rete blockchain privata e il raggiungimento degli obiettivi di business delle imprese e delle organizzazioni che saranno chiamate ad utilizzarla, sono state applicate un insieme di regole e di best-practices sia nella progettazione che nelle soluzioni proposte per la realizzazione della rete in funzione degli utilizzi previsti.
Nei seguenti capitoli è descritto in che modo è stato progettato ed implementato il sistema.

Qui di seguito è illustrato uno schema riassuntivo di quello che è lo scenario di utilizzo ipotizzato per la rete blockchain progettata e tutte le possibili movimentazioni di token all'interno della rete stessa.
I token coinvolti in tale scenario sono token di utilizzo. Come si può evincere, una delle principali caratteristiche del modello ipotizzato è che i token, una volta emessi, non vengano mai distrutti.
Le frecce nere rappresentano le transazioni di token tra i vari utilizzatori del sistema, mentre le frecce azzurre rappresentano i pagamenti in denaro (ad esempio l'acquisto dei token di utilizzo o eventuali pagamenti di commissioni per l'erogazione del servizio). Tali flussi sono puramente indicativi e saranno poi dettagliati all'interno del modello di business scelto per l'utilizzo della rete blockchain e dei servizi che verranno attivati.

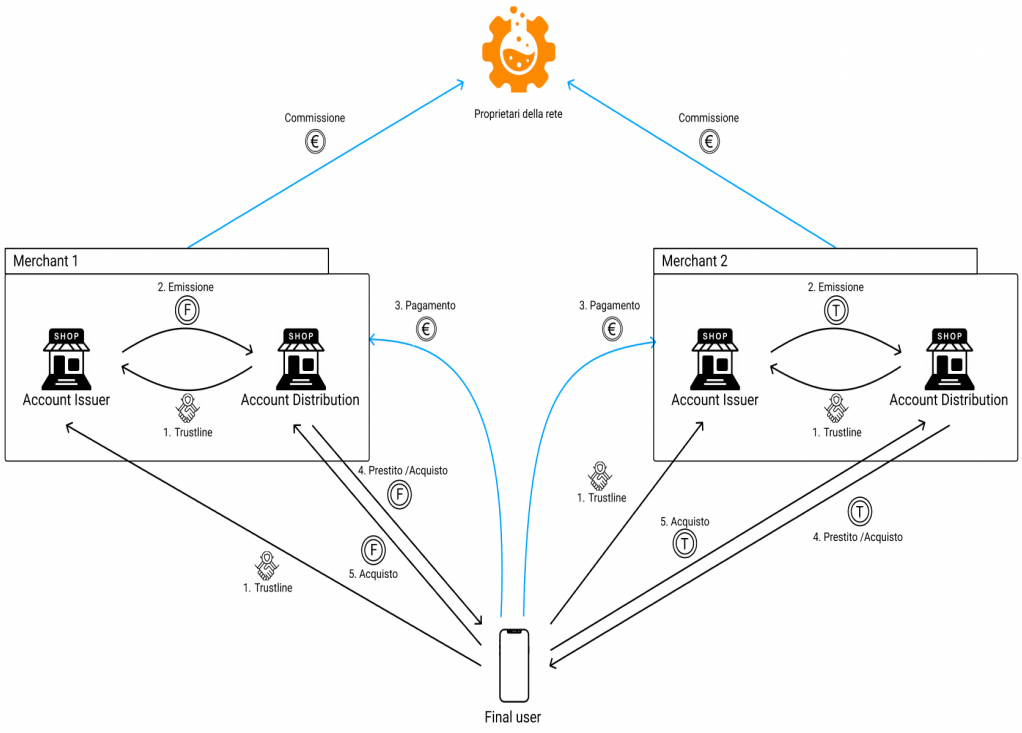


Figura 2.1: Rappresentazione scenario di utilizzo del sistema

## 2.2 Requisiti

Caratteristica prima e principale del progetto è la scalabilità. Non essendo presente un unico soggetto garante e centralizzato deve essere possibile, nel caso in cui si aggiunga un nuovo attore, poter integrare nuovi nodi all'interno della rete in qualsiasi momento senza compromettere la sicurezza del sistema. Per fare questo basterà configurare ed assegnare ai nuovi partecipanti fidati uno o più nodi per rendere ancora più distribuito e sicuro il processo di consenso. Il sistema, inoltre, dovrà essere sicuro e trasparente e le transazioni dovranno essere tracciabili e certificabili in ogni momento.
Il software BTKL, interfacciandosi con la rete realizzata, dovrà fornire le informazioni in modo sicuro solamente agli utilizzatori del sistema.
Il servizio di custody dovrà memorizzare in maniera crittografata tutte le chiavi private degli account, effettuando un mapping con la rispettiva chiave pubblica tra i vari account.
Chiunque voglia effettuare operazioni dovrà riuscire a firmare la transazione per poterla validare correttamente all'interno della rete. Per risalire alla chiave privata con cui firmare la transazione bisognerà prima autenticarsi attraverso il software BTKL.
Per poter comprendere ancora meglio gli obiettivi preposti ed il dominio del progetto sono stati definiti i confini e le caratteristiche del progetto.
La rete privata dovrà soddisfare i seguenti requisiti funzionali:

- Emissione e distribuzione di utility token da parte dei negozi affiliati.
- Gestione di pagamenti con utlity token P2P tra utenti finali.
- Monitoring delle transazioni effettuate.
- Monitoring del saldo del proprio account.

I requisiti funzionali descritti, per comprendere meglio il dominio dell'elaborato, sono tradotti nel seguente diagramma dei casi d'uso.

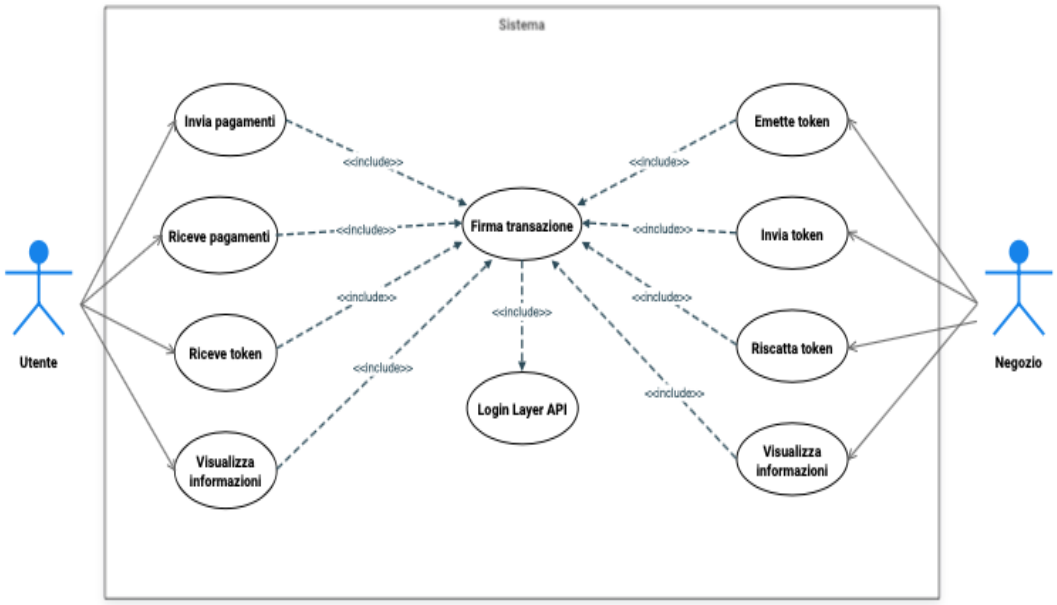


Figura 2.2: Diagramma dei casi d'uso

Utente e Negozio sono i due attori rappresentati nel diagramma poichè sono coloro che partecipano attivamente tramite i propri account allo scambio di token nella rete. Il negozio emette e distribuisce gli utility token ai vari clienti. L'utente è il singolo cliente che può acquistare e successivamente spendere i token emessi dal negozio in cambio di beni materiali o può decidere di scambiare i token con altri utenti. La firma delle transazioni è il passaggio che permette la validazione di qualsiasi azione nella rete privata. Gli attori con i propri nodi e gli altri componenti software non sono stati rappresentati all'interno di questo diagramma in quanto non svolgono le funzionalità descritte in maniera attiva.

# Capitolo 3

# Progettazione

## 3.1 Architettura infrastruttura

L'infrastruttura realizzata in grado di ospitare una blockchain privata basata su Stellar, è formata da 5 elementi primari :

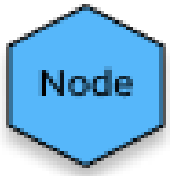


Figura 3.1: **Nodo Full Validator**, rappresenta il vero e proprio nodo della rete Stellar. Ha il compito di sottomettere le transazioni scrivendole nel registro; inoltre valida e pubblica lo stato di quest'ultimo su un archivio storico.


Figura 3.2: **Nodo Horizon**, è un nodo che mette a disposizione API che permettono di interagire con la rete Stellar.


Figura 3.3: **BTK Layer**, è il componente software che comunica con l'istanza Horizon. Qui saranno create API ad-hoc per restituire dei dati conformi alle esigenze degli utilizzatori.

Figura 3.4: **Database di custody**, è il database in cui verranno salvate le chiavi pubbliche e private degli utenti.

Figura 3.5: **Load Balancer**, è un software in grado di smistare le richieste, siano esse transazioni o semplici richieste per analizzare i dati, e assegnare il compito alle molteplici istanze del BTK Layer. È stato scelto Nginx come sistema di load balancing.

### Ipotesi di flusso

Si ipotizzi, per comprendere al meglio l'utilità delle varie componenti, il flusso attraverso il quale un utente (Alice) sottomette una transazione al sistema. Alice effettua l'acquisto di un prodotto in un negozio convenzionato utilizzando utility token, attraverso l'app mobile ufficiale.

- La richiesta della transazione viene trasmessa dall'app al sistema di Load Balancer, il quale la indirizzerà al server con il nodo Horizon più scarico in quel momento. Per la precisione, la richiesta viene inviata al BTK Layer installato sullo stesso server in cui si trova il nodo Horizon.
- Il BTK Layer comporrà la transazione prendendo i dati ricevuti, contatterà il Database di Custody per poter recuperare ed utilizzare le chiavi necessarie per firmare la transazione ed infine invierà la richiesta al Nodo Istanza Horizon.
- L'istanza Horizon, trasmetterà la transazione ai Nodi Full Validator i quali si occuperanno di validare e scrivere la transazione nel registro.
- Una volta chiuso il registro, i dati della transazione completata saranno trasmessi dai nodi Full Validator al nodo Istanza Horizon, il quale invierà nuovamente l'informazione al BTK Layer.
- A quel punto, il BTK Layer formatterà il dato di risposta e lo trasmetterà indietro al Load Balancer, che a sua volta lo invierà all'app dove è stata eseguita la transazione.

**Bilanciamento server**

Partendo da questi 5 elementi, sono stati delineati 4 tipi di server:

- **Server con Nodo Full Validator**: per garantire una maggior persistenza e duplicazione del registro, si raccomanda l'utilizzo di almeno 3 server.

- **Server con Nodo Istanza Horizon e BTK Layer**: i due elementi primari (Nodo istanza Horizon e BTK Layer) saranno installati sul medesimo server. Al fine di garantire la corretta operatività e ridondanza dell'infrastruttura, si consiglia l'utilizzo di almeno 2 server.

- **Server con Database di Custody**: al fine di garantire una maggior sicurezza inerente il repository contenente tutte le chiavi (sia pubbliche che private) degli utenti, si è scelto di utilizzare un server appositamente dedicato a tale scopo.

- **Server con Load Balancer**: in questo server sarà installato il sistema di smistamento delle richieste provenienti dagli utenti; questo server rappresenterà quindi il punto di accesso all'infrastruttura. In una prima fase, sarà possibile dedicare un solo server allo svolgimento di tale attività, successivamente invece potrà delinearsi la necessità di strutturare meccanismi di ridondanza, atti ad eliminare eventuali criticità operative. In questo caso sarà necessario installare almeno un altro server dedicato a tale scopo.

In totale quindi, il numero minimo di server necessari al corretto funzionamento della infrastruttura proposta, è di 7 unità:

- 3 server "Nodo Full Validator"

- 2 server "Nodo Horizon + BTK Layer"

- 1 server "Database Custody"

- 1 server "Load Balancer"

L'ipotesi di dimensionamento dei Server necessari al corretto funzionamento dell'infrastruttura, è stata effettuata mediante lo svolgimento di stress test su macchine AWS EC2 in cloud. Per lo svolgimento dei test, si è provveduto a ricostruire per 4 volte l'intera infrastruttura, utilizzando differenti tipologie di macchine in termini di risorse cpu e ram. È stato poi creato un meccanismo automatico per la sottomissione di transazioni, al fine di simulare l'eventuale attività svolta dagli utenti. Per ognuna delle 4 differenti infrastrutture (in termini di risorse disponibili), sono state sottomesse 4 differenti fasce di transazioni, così da poter ottenere una tabella comparativa in grado di fornire informazioni utili alla determinazione delle risorse necessarie al corretto funzionamento dell'infrastruttura. Le fasce di transazioni sottomesse al sistema sono state:

- 500 tx/5s
- 1000 tx/5s
- 2000 tx/5s
- 4000 tx/5s

Si ricorda che il termine tx/5s indica il numero di transazioni sottomesse e validate dai nodi della blockchain ogni 5 secondi. La blockchain di Stellar è stata sviluppata in modo da chiudere un registro ogni 5 secondi. Ad ogni chiusura di registro verranno pertanto eseguite tutte le transazioni validate e scritte nei 5 secondi precedenti. Parlando quindi 2000 tx/5s, si ipotizza che ci siano 2.000 utenti che, nell'arco di 5 secondi, sottomettono contemporaneamente una transazione alla piattaforma.
Il primo dato interessante scaturito dai test è quello per cui il numero di transazioni sottomesse al sistema non ha avuto alcun impatto rilevante né sul server dedicato al Database di custody, né su quello dedicato al Load Balancer. Per questi due server quindi, si decide per l'utilizzo di *Virtual Private Server* (VPS) con risorse pari a quelle di una macchina AWS m5.xlarge, ossia un server con 4 cpu e 16 GB di ram.
Per quanto riguarda i server dedicati ai nodi Full Validator ed Horizon, il discorso invece è nettamente differente.

Questi ultimi rappresentano coloro che devono prendere in carico l'elaborazione delle informazioni e la loro attività è strettamente collegata al numero di transazioni che vengono sottomesse all'infrastruttura.
Nella tabella sotto riportata, si può notare per ognuno dei quattro test, le risorse (in termini di cpu e ram) di ogni macchina dedicata a ricoprire il ruolo di nodo Full Validator piuttosto che nodo Horizon.

| | Configurazione TEST 1 | | Configurazione TEST 2 | | Configurazione TEST 3 | | Configurazione TEST 4 | |
|---|---|---|---|---|---|---|---|---|
| **Nodo** | **Horizon** | **Full Validator** | **Horizon** | **Full Validator** | **Horizon** | **Full Validator** | **Horizon** | **Full Validator** |
| **Modello EC2** | m5.xlarge | m5.large | m5.2xlarge | m5.large | m5.2xlarge | m5.xlarge | m5.4xlarge | m5.xlarge |
| **N° CPU** | 4 | 2 | 8 | 2 | 8 | 4 | 16 | 4 |
| **RAM (GB)** | 16 | 8 | 32 | 8 | 32 | 16 | 64 | 16 |

Figura 3.6: Tabella configurazione test

Nella tabella seguente invece, si riportano i risultati inerenti il raffronto tra i 4 test e le 4 fasce di transazioni sottomesse alla piattaforma.
Si può notare che nel caso si decidesse di optare per un'infrastruttura in grado di gestire 2000 tx/5s, si dovranno utilizzare le seguenti tipologie di server:

- Nodo Full Validator : 2 CPU e 8GB di RAM
- Nodo Horizon : 16 CPU e 64 GB di RAM

| | | Configurazione TEST 1 | | Configurazione TEST 2 | | Configurazione TEST 3 | | Configurazione TEST 4 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Horizon | Full Validator | Horizon | Full Validator | Horizon | Full Validator | Horizon | Full Validator |
| **500** tx/5s | CPU | 40% | 9% | 22% | 13% | 22% | 5% | 13% | 5% |
| | RAM | 11% | 6% | 6% | 9% | 6% | 3% | 3% | 3% |
| **1000** tx/5s | CPU | 60% | 13% | 55% | 24% | 55% | 13% | 21% | 10% |
| | RAM | 18% | 9% | 10% | 9% | 10% | 4% | 4% | 4% |
| **2000** tx/5s | CPU | 96% | 35% | 74% | 30% | 74% | 20% | 50% | 17% |
| | RAM | 32% | 10% | 15% | 9% | 15% | 5% | 7% | 4% |
| **4000** tx/5s | CPU | / | / | 100% | 43% | 100% | 30% | 61% | 30% |
| | RAM | / | / | 24% | 8% | 24% | 4% | 12% | 4% |

Figura 3.7: Tabella carico di utilizzo

## 3.2 Soluzioni proposte

### Soluzione cloud

La prima soluzione, come mostrato in Figura 3.8, prevede la configurazione di 7 istanze EC2 in cloud su AWS, nel dettaglio:

- 3 istanze EC2 dedicate ciascuna all'esecuzione dei nodi Full Validator.
- 2 istanze EC2 ognuna dedicata ad un nodo istanza Horizon e del BTK Layer.
- 1 istanza EC2 dedicata unicamente al database di custody.
- 1 istanza EC2 dedicata all'esecuzione del servizio di load balancer Nginx.

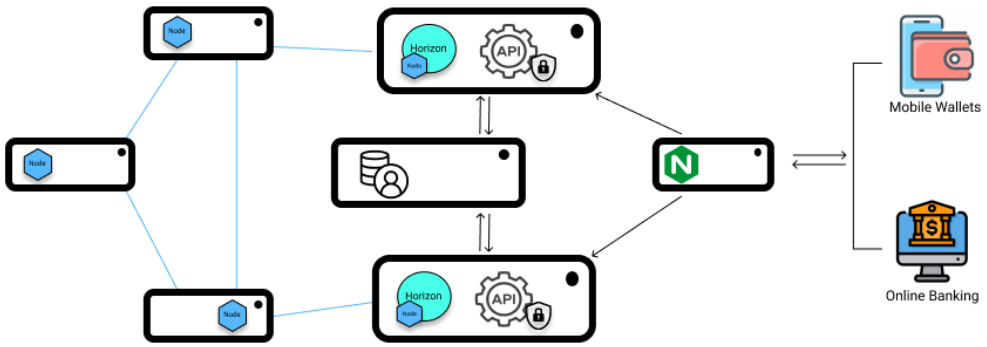


Figura 3.8: Soluzione 1: cloud

Il vantaggio principale è quello per cui il cloud può rappresentare un'interessante soluzione per partire in tempi brevi e con poco sforzo economico nel caso non si disponga di risorse nella propria infrastruttura interna. È quindi una soluzione rapida, semplice ed economica.
Tuttavia è la configurazione meno sicura tra quelle proposte a causa del fatto che l'intera infrastruttura non risiede in un data center controllato ma è cloud su server Amazon.

### Soluzione on premise

In questa soluzione viene ripresa la stessa struttura definita nella soluzione precedente. In questo caso vengono configurate 7 VPS o macchine fisiche all'interno del data center.
Nel proporre questa soluzione, si ipotizza quindi che si disponga di un data center di proprietà in cui siano disponibili le risorse necessarie al deploy dell'intera infrastruttura.
I vantaggi di questa soluzione sono:

- Nel caso siano disponibili le risorse all'interno del proprio data center, i tempi di deploy equivalgono a quelli della soluzione precedente.
- I dati sono dislocati *on-premise* e quindi protetti all'interno del perimetro aziendale.

Nel caso in cui non si disponga già delle risorse necessarie, si dovrà provvedere alla loro implementazione con un conseguente aumento delle tempistiche.

### Soluzione iperconvergente

Nell'ultima soluzione proposta, si prende in esame l'ipotesi di realizzare un'infrastruttura iperconvergente installata presso il data center di proprietà e dedicarlo esclusivamente alla piattaforma Stellar.
In questa infrastruttura saranno quindi realizzati i 7 server necessari all'implementazione della blockchain privata, come indicato nelle soluzioni precedenti; con la possibilità di estendere il numero di nodi e le loro risorse in funzione di quelle rese disponibili dall'infrastruttura stessa.
Un' infrastruttura iperconvergente è un modello software-defined, che fornisce risorse di calcolo, memorizzazione, networking e virtualizzazione, prendendole da apparati hardware da essa gestiti.
I principali vantaggi di una piattaforma di questo tipo sono dati dalla grande scalabilità delle risorse messe a disposizione e dagli elevati livelli di sicurezza. In particolare vi è garanzia di continuità dei servizi ed un'ottima gestione del *disaster recovery.*
Una piattaforma iperconvergente è quindi formata da una serie di apparati hardware (server che mettono a disposizione cpu, ram, hard disk, networking,

etc..), gestiti da un layer software che è in grado di unificare le loro risorse per renderle disponibili alla fruizione, attraverso la creazione di Virtual Private Server.

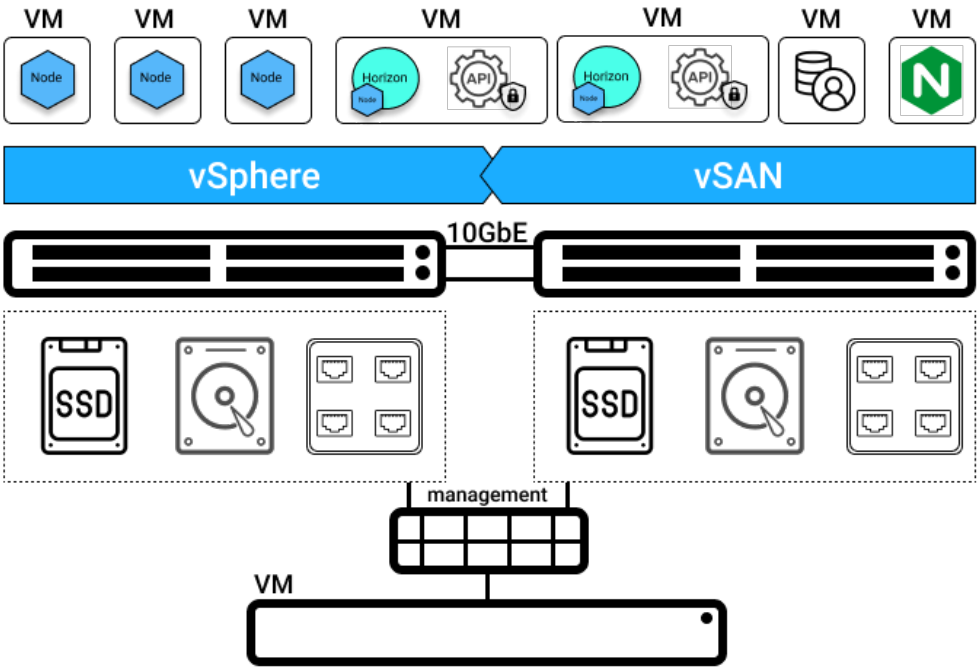


Figura 3.9: Soluzione 3: iperconvergenza

In questa soluzione, si propone l'acquisto di 2 macchine fisiche collegate tra loro in fibra ottica 10 Gbit, unitamente a tutti gli elementi software necessari alla corretta gestione dell'iperconvergenza.
Ogni macchina avrà la capacità di eseguire al suo interno tutti i moduli software della blockchain privata, garantendo una notevole scalabilità sia nel numero dei nodi sia delle risorse disponibili ad ogni nodo. L'alta affidabilità delle risorse software sarà garantita dall'infrastruttura iperconvergente, in grado di migrare i server virtuali da un server fisico all'altro a fronte di un eventuale guasto.
Posizionando inoltre i due nodi fisici in differenti data center, mantenendoli sempre connessi con fibra a 10 Gbit, si potrebbe ottenere una prima soluzione di disaster recovery dell'intera infrastruttura.

I vantaggi che derivano dalla realizzazione di questa infrastruttura sono i seguenti:

- elevata tolleranza ai guasti.
- possibilità di realizzare una prima soluzione di disaster recovery.
- eliminazione di ogni Single Point Of Failure.
- maggior scalabilità in termini di risorse disponibili per l'implementazione di nuovi nodi e/o l'espansione di quelli già esistenti.

Questa soluzione risulta essere tuttavia la più complessa e quella per cui è richiesto uno sforzo economico maggiore.

## 3.3 BTK Layer - BTKL

Il software BTK Layer è stato creato per rispondere a due principali necessità nell'ecosistema architetturale:

- **salvare le chiavi e comunicare con il database di custody**: nell'attuale struttura dell'architettura BTKL è l'unico canale comunicante con il database di custody. Esso gestisce il salvataggio delle chiavi al momento della creazione di un nuovo account e le recupera in caso di necessità. In tale maniera è possibile gestire operazioni complesse restituendo endpoint semplici all'interfaccia di utilizzo.

- **semplificare l'utilizzo e la comunicazione con la blockchain privata**: la blockchain Stellar mette a disposizione delle API atomiche, tramite Horizon, che svolgono compiti ben precisi. La combinazione di queste API comporta la realizzazione di qualcosa di concreto, come può essere per esempio la creazione di un utente o di un asset. Bisogna eseguirle in un preciso ordine e bisogna essere in possesso di tutte le informazioni necessarie per completarle con successo. Il software BTKL si interpone fra le API fornite da Horizon e il canale di comunicazione con i software destinati ad essere utilizzati dagli utenti finali. Il suo scopo è quello di fornire una serie di endpoint in grado di svolgere molteplici azioni necessarie alla realizzazione di una richiesta complessa, come può essere la creazione di un asset, la distribuzione degli asset tra gli utenti o la visualizzazione dello storico delle transazioni eseguite da un dato utente. Il software BTKL offrirà infatti la possibilità di elaborare e analizzare i dati scritti all'interno della rete blockchain, restituendoli in maniera già aggregata all'interfaccia predisposta per l'utilizzo lato utente.

Per poter svolgere tutte le funzionalità richieste il layer BTK, in seguito ad un processo di autenticazione, esporrà le seguenti RESTful API:

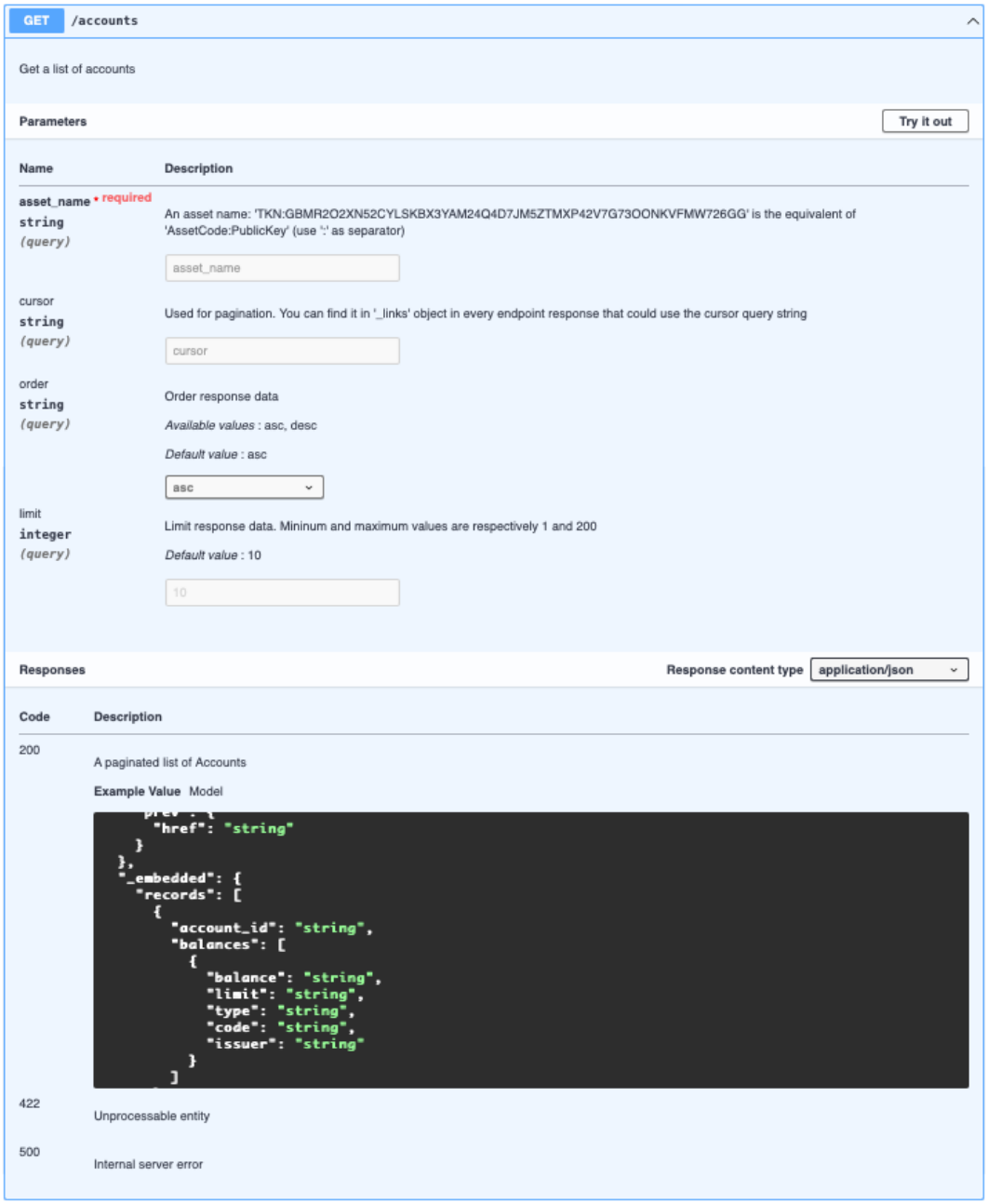


Figura 3.10: GET/accounts

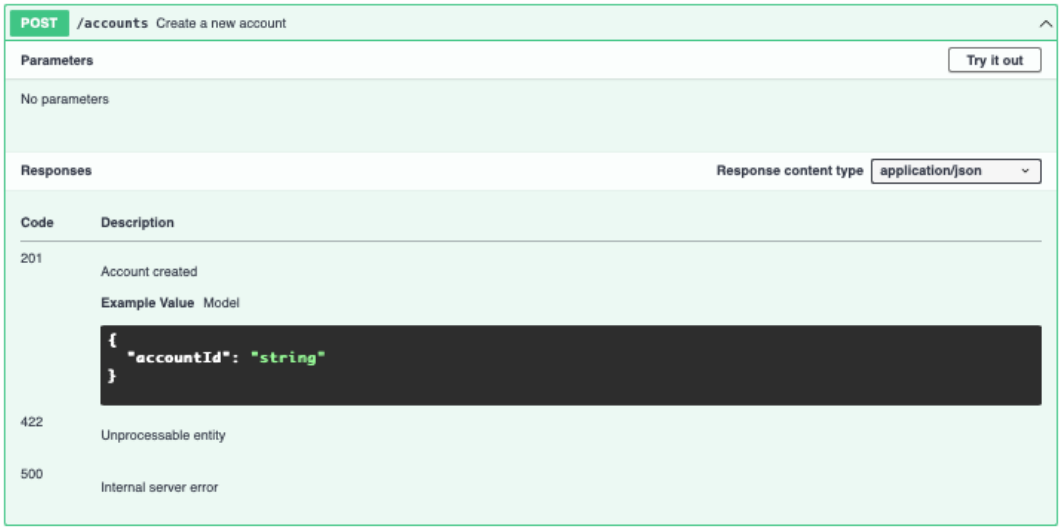


Figura 3.11: POST/accounts

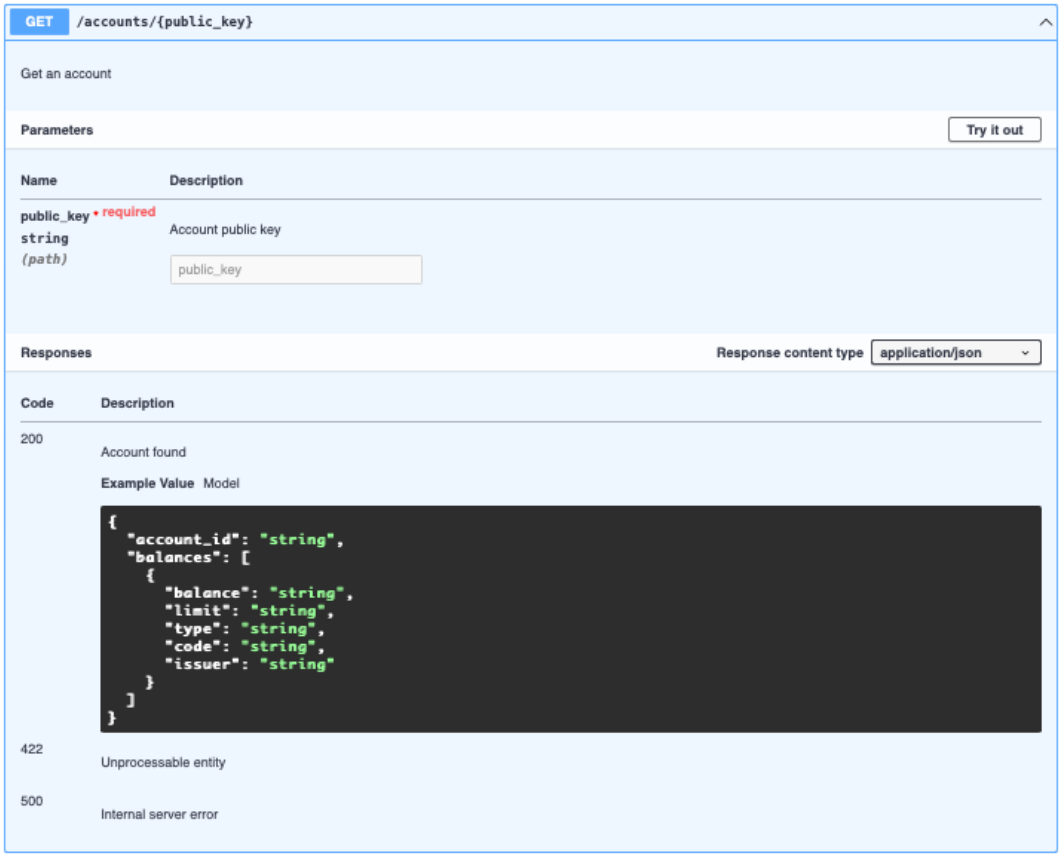


Figura 3.12: GET/accounts/{public_key}

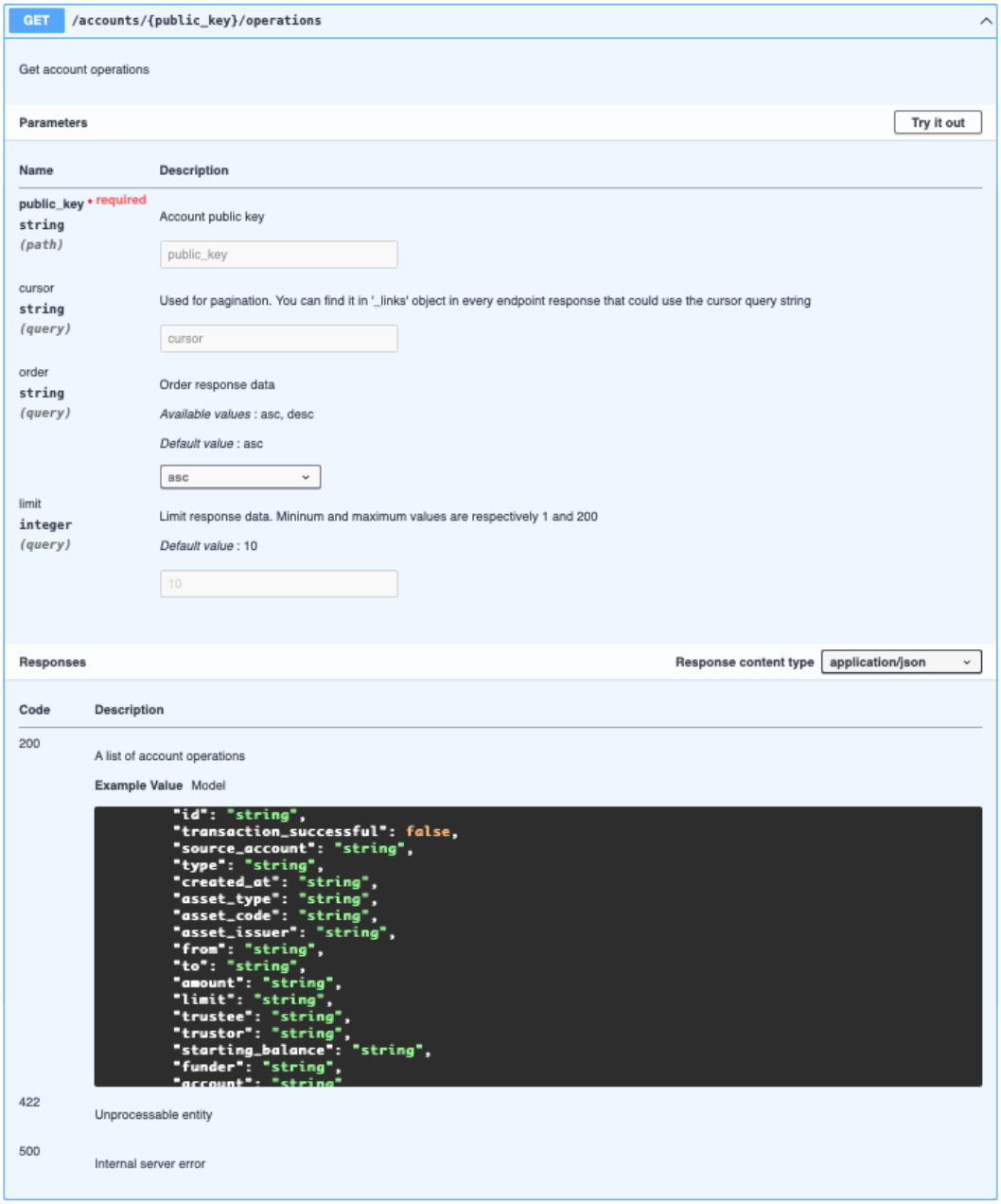


Figura 3.13: GET/accounts/{public_key}/operations

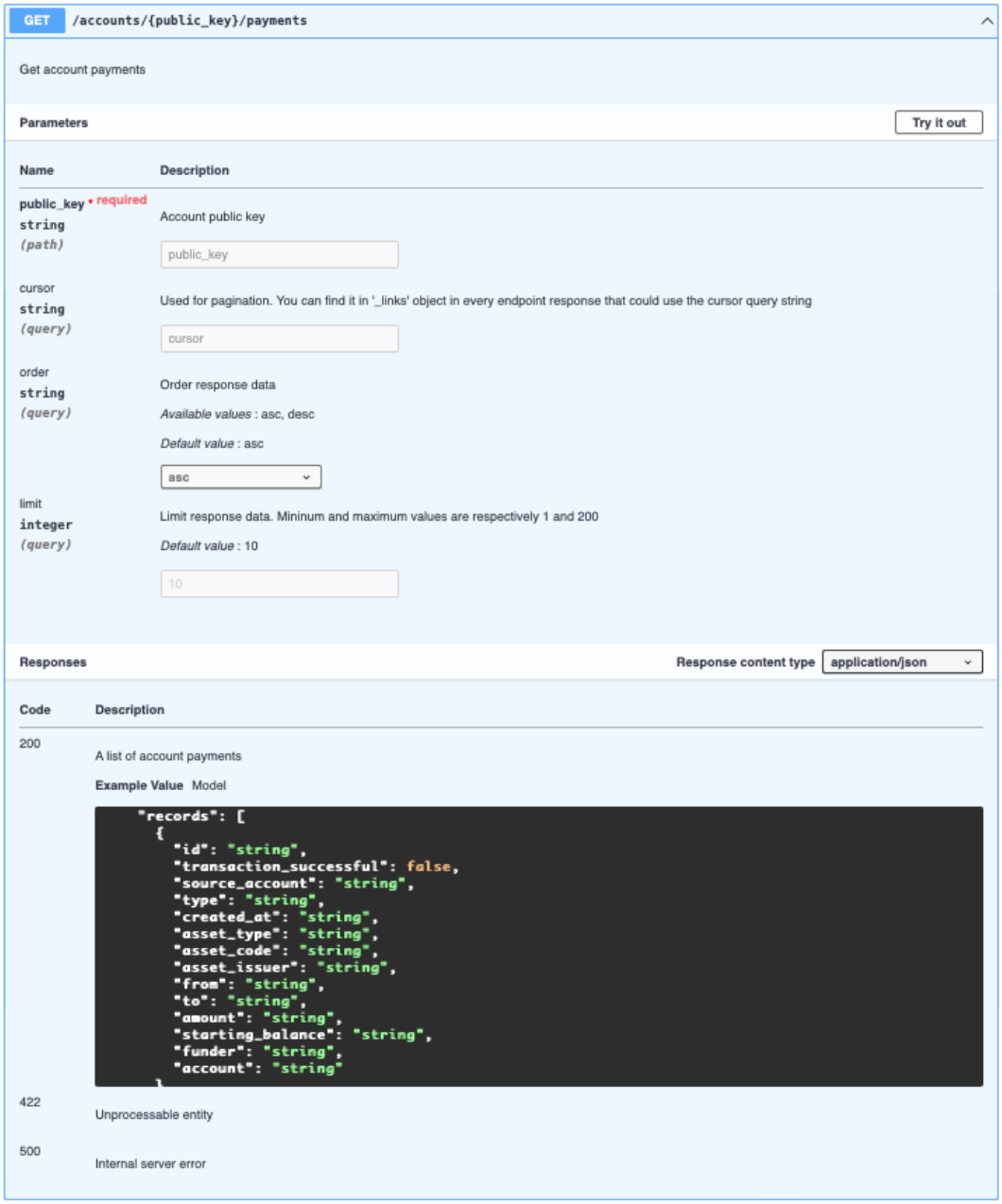


Figura 3.14: GET/accounts/{public_key}/payments

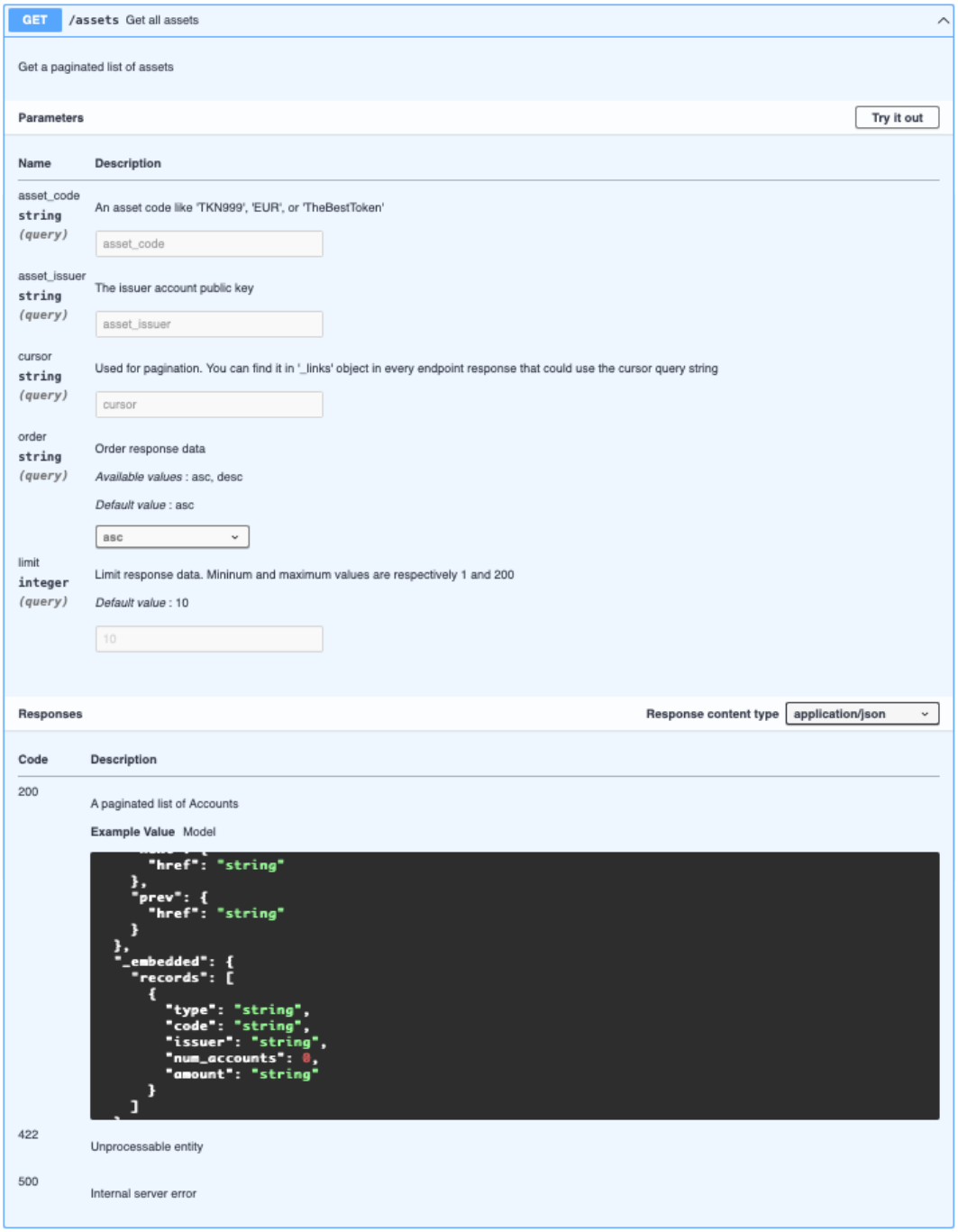


Figura 3.15: GET/assets

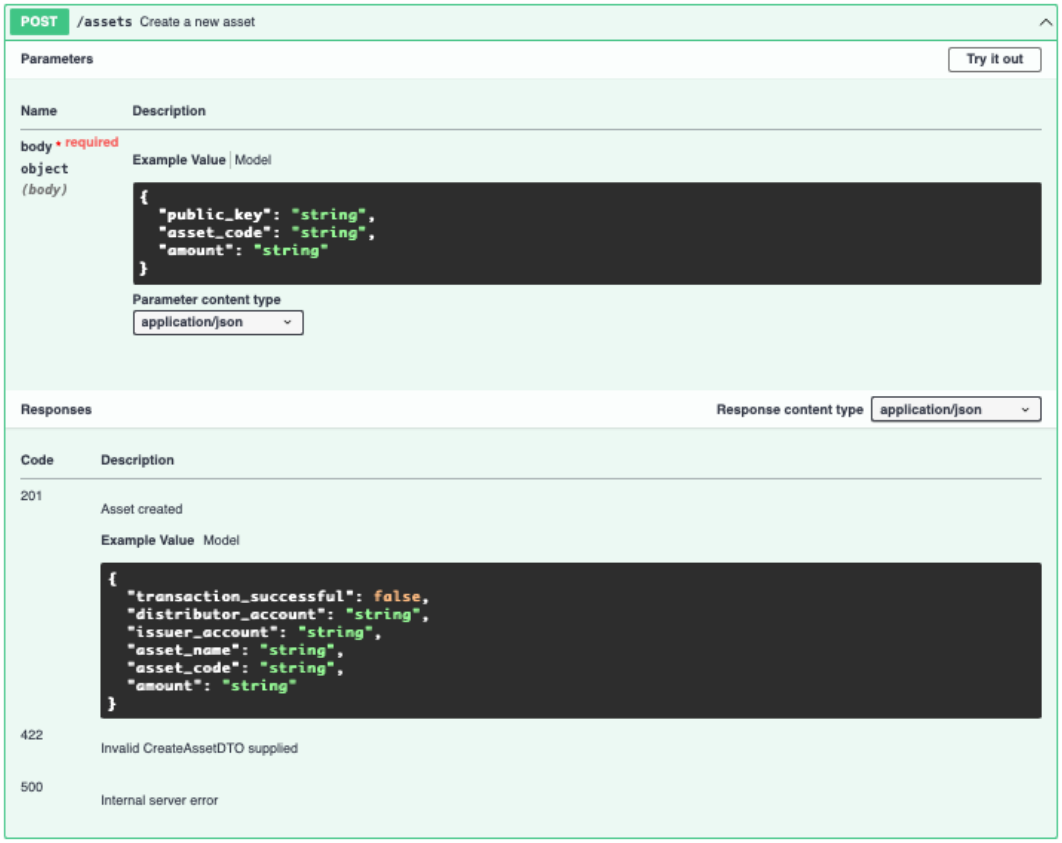


Figura 3.16: POST/assets

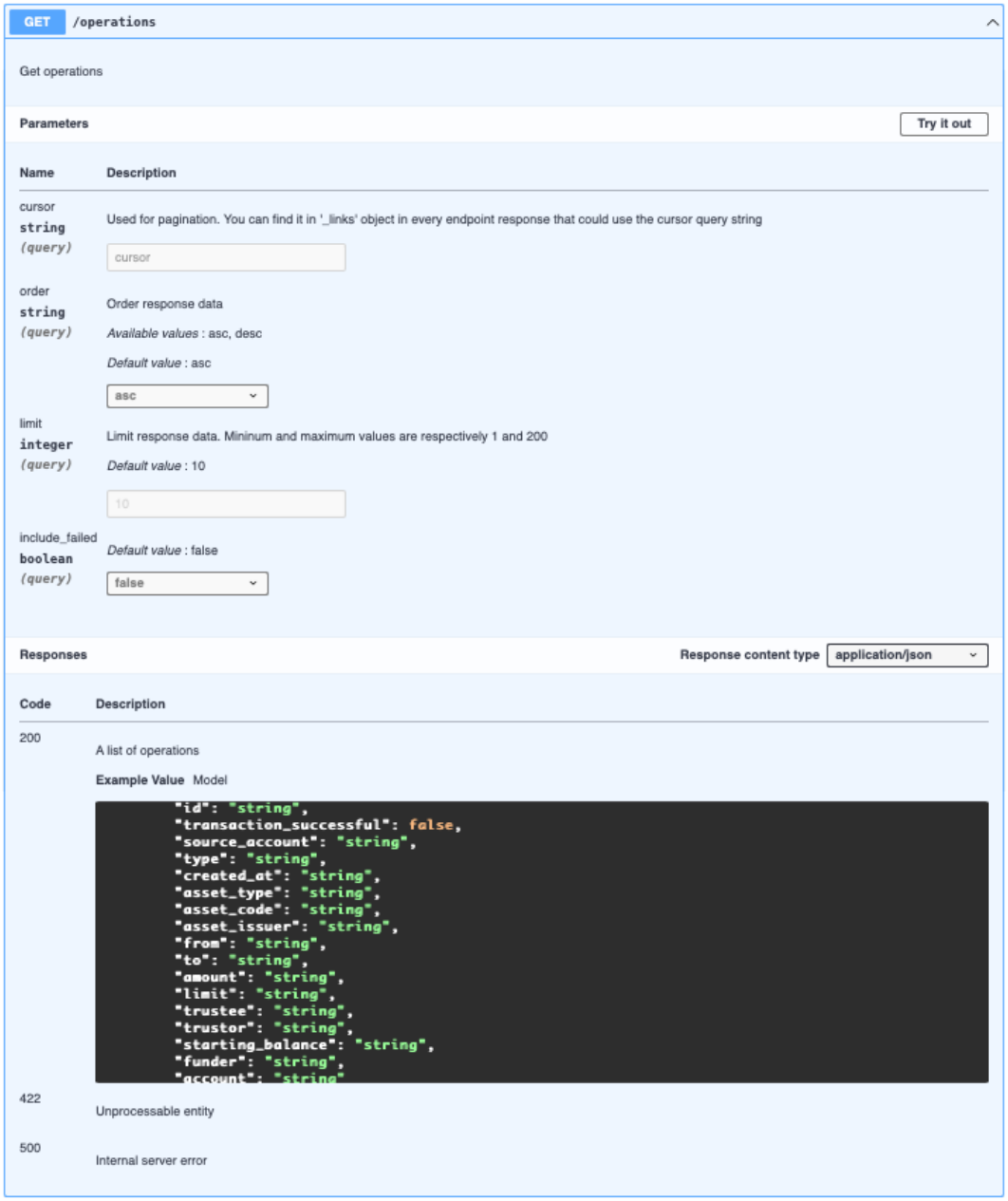


Figura 3.17: GET/operations

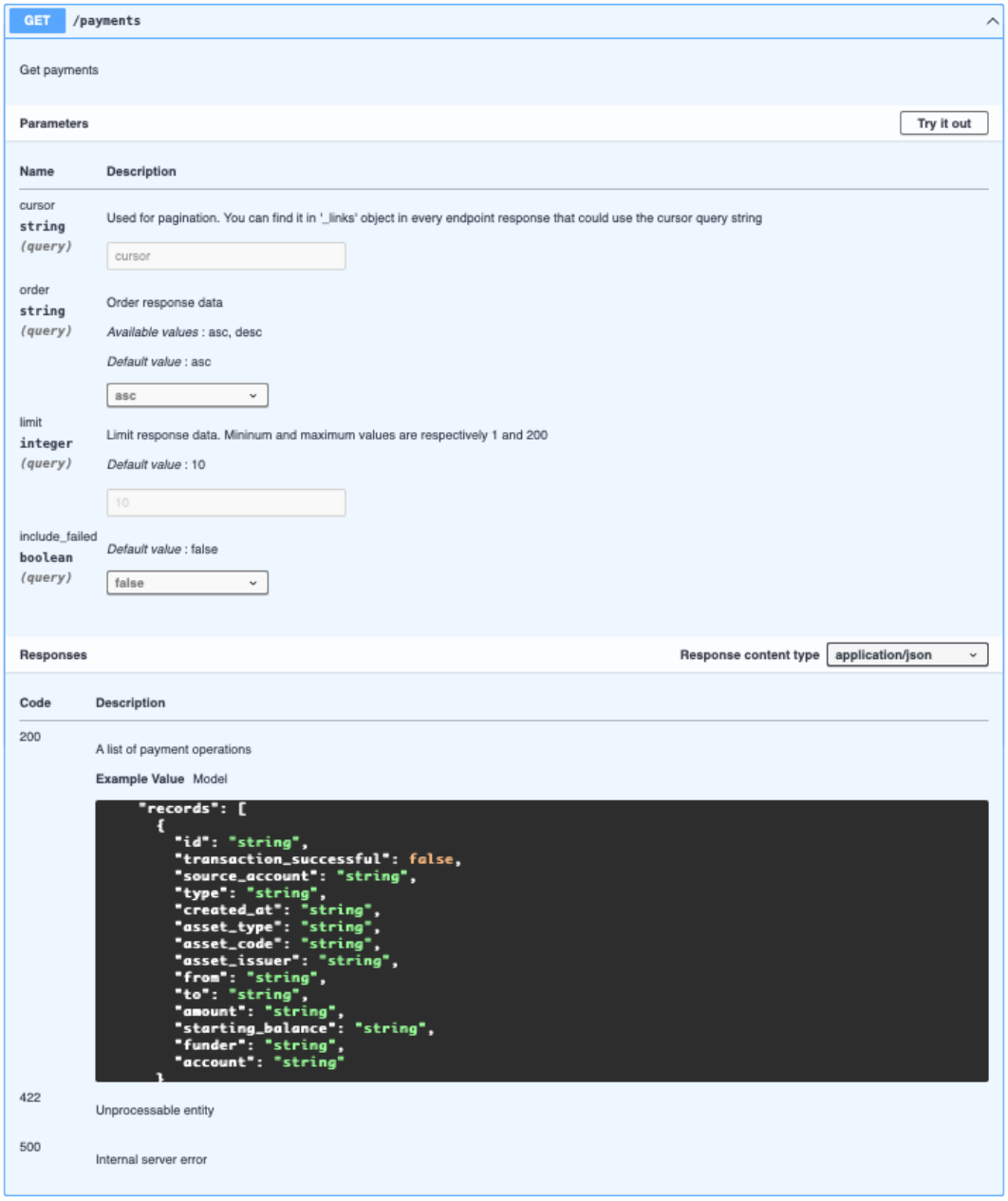


Figura 3.18: GET/payments

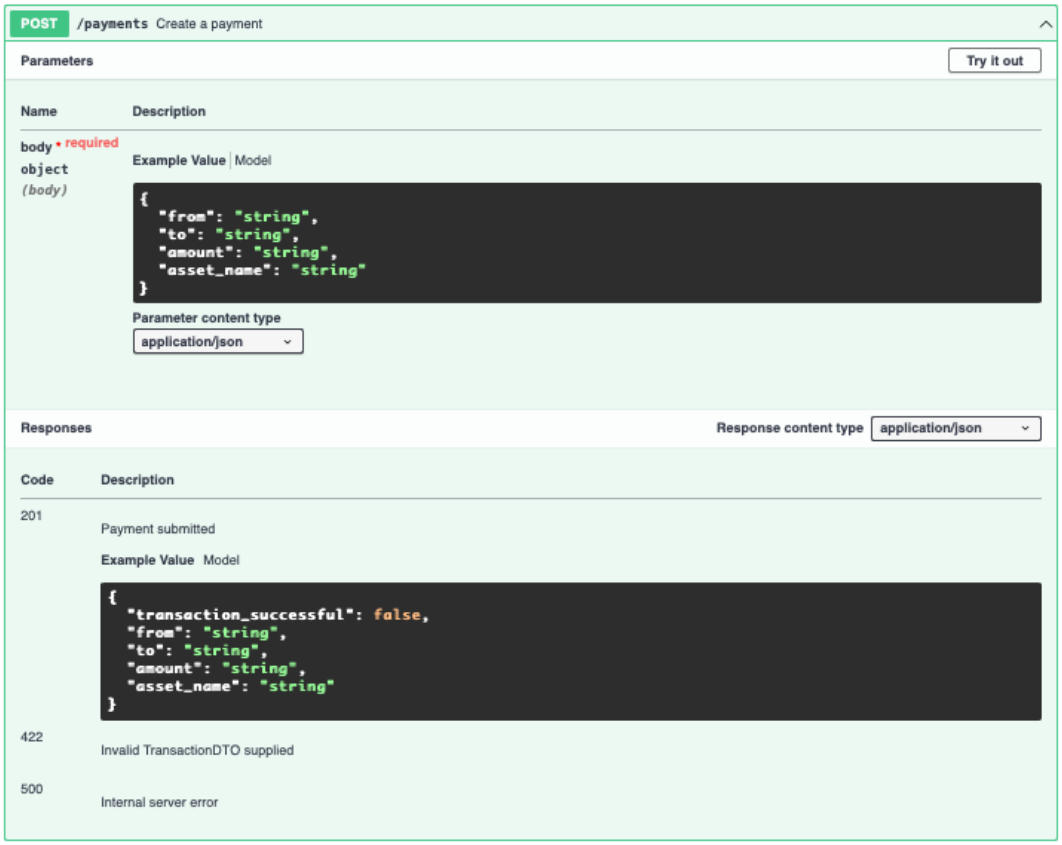


Figura 3.19: POST/payments

# Capitolo 4

# Implementazione

## 4.1 Nodi Stellar

Per poter eseguire tutti i test descritti nel capitolo precedente e testare la reale fattibilità del progetto, si è deciso di implementare inizialmente la struttura di base della rete: 3 nodi Full Validator ed un nodo Horizon.

### Nodo Full Validator

Nelle pagine seguenti viene mostrato uno dei file di configurazione utilizzato per la creazione di un nodo Full Validator di Stellar.

```
LOG_FILE_PATH=""
BUCKET_DIR_PATH="/opt/stellar/buckets"
DATABASE="postgresql://dbname=core host=ec2-52-15-193-153.us-east-2.compute
    .amazonaws.com_db user=stellar password=******"
NETWORK_PASSPHRASE="My Network ; February 2022"
NODE_SEED="SCOGNX3OT37HQFX4MAO3ZYM7W4E2WYCOHB5QCUX66MCRUGF6WDX7SXWQ"
NODE_HOME_DOMAIN="botika.ai"
METADATA_OUTPUT_STREAM=""

LOG_COLOR=false
ENTRY_CACHE_SIZE=100000
PREFETCH_BATCH_SIZE=1000
HTTP_PORT=11626
PUBLIC_HTTP_PORT=true
```

```
14 HTTP_MAX_CLIENT=128
15 PEER_PORT=11625
16 TARGET_PEER_CONNECTIONS=8
17 MAX_ADDITIONAL_PEER_CONNECTIONS=-1
18 MAX_PENDING_CONNECTIONS=500
19 PEER_AUTHENTICATION_TIMEOUT=2
20 PEER_TIMEOUT=30
21 PEER_STRAGGLER_TIMEOUT=120
22 MAX_BATCH_WRITE_COUNT=1024
23 MAX_BATCH_WRITE_BYTES=1048576
24 FLOOD_OP_RATE_PER_LEDGER=1.0
25 FLOOD_TX_PERIOD_MS=200
26 PREFERRED_PEERS_ONLY=false
27 MINIMUM_IDLE_PERCENT=0
28 NODE_IS_VALIDATOR=true
29 FAILURE_SAFETY=-1
30 UNSAFE_QUORUM=false
31 CATCHUP_COMPLETE=false
32 CATCHUP_RECENT=0
33 WORKER_THREADS=11
34 QUORUM_INTERSECTION_CHECKER=true
35 MAX_CONCURRENT_SUBPROCESSES=16
36 AUTOMATIC_MAINTENANCE_PERIOD=359
37 AUTOMATIC_MAINTENANCE_COUNT=400
38 AUTOMATIC_SELF_CHECK_PERIOD=10800
39 RUN_STANDALONE=false
40 MANUAL_CLOSE=false
41 ARTIFICIALLY_GENERATE_LOAD_FOR_TESTING=false
42 ARTIFICIALLY_ACCELERATE_TIME_FOR_TESTING=false
43 ARTIFICIALLY_SET_CLOSE_TIME_FOR_TESTING=0
44 ALLOW_LOCALHOST_FOR_TESTING=false
45 CATCHUP_WAIT_MERGES_TX_APPLY_FOR_TESTING=false
46 MAXIMUM_LEDGER_CLOSETIME_DRIFT=50
47 DISABLE_XDR_FSYNC=false
48 MAX_SLOTS_TO_REMEMBER=12
49 EXPERIMENTAL_PRECAUTION_DELAY_META=false
50 METADATA_DEBUG_LEDGERS=0
51
52 COMMANDS=[]
53 NODE_NAMES=[]
54 PREFERRED_PEERS=[]
```

```
55 PREFERRED_PEER_KEYS=[]
56 SURVEYOR_KEYS=[]
57 KNOWN_PEERS=[]
58 KNOWN_CURSORS=[]
59 INVARIANT_CHECKS=[]
60 EXCLUDE_TRANSACTIONS_CONTAINING_OPERATION_TYPE=[]
61
62 [[HOME_DOMAINS]]
63 HOME_DOMAIN="botika.ai"
64 QUALITY="HIGH"
65
66
67 [[VALIDATORS]]
68 NAME="ec2-13-59-185-133.us-east-2.compute.amazonaws.com"
69 HOME_DOMAIN="botika.ai"
70 PUBLIC_KEY="GDJG4VI5K5RQ64JRTPGMBMLZQTKQTBG6UFEJ6JI2N23S5WU4R4KIWRC7"
71 ADDRESS="ec2-13-59-185-133.us-east-2.compute.amazonaws.com"
72 HISTORY="curl -sf ec2-13-59-185-133.us-east-2.compute.amazonaws.com/vs/{0}
       -o {1}"
73
74 [[VALIDATORS]]
75 NAME="ec2-18-219-138-173.us-east-2.compute.amazonaws.com"
76 HOME_DOMAIN="botika.ai"
77 PUBLIC_KEY="GADN2ABIAH6IKFLGWJX2WWCHOTDJQVUVGGA3OVOQUVAQAD3XFNAEQM7R"
78 ADDRESS="ec2-18-219-138-173.us-east-2.compute.amazonaws.com"
79 HISTORY="curl -sf ec2-18-219-138-173.us-east-2.compute.amazonaws.com/vs/{0}
        -o {1}"
80
81
82 [HISTORY.vs]
83 get="cp /opt/stellar/history/vs/{0} {1}"
84 put="cp {0} /opt/stellar/history/vs/{1}"
85 mkdir="mkdir -p /opt/stellar/history/vs/{0}"
```

Nelle prime 7 righe del codice proposto sono raccolte informazioni riguardo il nodo che si sta configurando. Vi è specificato infatti a quale dominio appartiene e a quale rete l'istanza deve far riferimento. Inoltre viene definita la connessione con il database e indicato l'indirizzo dove memorizzare i bucket. A riga 28 è possibile notare la variabile *NODE_IS_VALIDATOR* impostata a true. Questa operazione è stata necessaria perchè per impostazione predefinita, i nodi Stellar

non sono impostati per la validazione. Se si desidera che il proprio nodo sia un Basic Validator o un Full Validator, bisogna configurarlo esplicitamente. Per concludere da riga 62 a riga 79 viene definito il quorum set. Dopo aver specificato la qualità del dominio e aggiunto i validatori alla tua configurazione, Stellar genera automaticamente un quorum set utilizzando le seguenti regole:

- I validatori con lo stesso dominio vengono automaticamente raggruppati e dotati di una soglia che richiede la maggioranza semplice (2f+1)
- A gruppi eterogenei di validatori viene assegnata una soglia presupponendo un fallimento bizantino (3f+1)

### Nodo Horizon

Oltre all'implementazione dei nodi validatori è stato necessario configurare almeno un nodo Horizon per poter accedere agilmente ai dati della rete privata.
Mentre un nodo validatore Stellar richiede una configurazione complessa con molti campi e variabili, il file di configurazione di Horizon può essere mantenuto estremamente semplice. La maggior parte della configurazione verrà generata automaticamente in fase di esecuzione.
Nell'esempio che segue il nodo Horizon si collega a 3 nodi Full Validator della nostra rete, tra i quali, come si può notare anche al nodo dell'esempio precedente.

```
UNSAFE_QUORUM=false
FAILURE_SAFETY=-1
HTTP_PORT=11626
PEER_PORT=11725

NETWORK_PASSPHRASE="My Network ; February 2022"

[[HOME_DOMAINS]]
HOME_DOMAIN="botika.ai"
QUALITY="MEDIUM"


[[VALIDATORS]]
NAME="ec2-13-59-185-133.us-east-2.compute.amazonaws.com"
HOME_DOMAIN="botika.ai"
PUBLIC_KEY="GDJG4VI5K5RQ64JRTPGMBMLZQTKQTBG6UFEJ6JI2N23S5WU4R4KIWRC7"
ADDRESS="ec2-13-59-185-133.us-east-2.compute.amazonaws.com"
HISTORY="curl -sf ec2-13-59-185-133.us-east-2.compute.amazonaws.com/vs/{0}
    -o {1}"

[[VALIDATORS]]
NAME="ec2-18-219-138-173.us-east-2.compute.amazonaws.com"
HOME_DOMAIN="botika.ai"
PUBLIC_KEY="GADN2ABIAH6IKFLGWJX2WWCHOTDJQVUVGGA3OVOQUVAQAD3XFNAEQM7R"
ADDRESS="ec2-18-219-138-173.us-east-2.compute.amazonaws.com"
HISTORY="curl -sf ec2-18-219-138-173.us-east-2.compute.amazonaws.com/vs/{0}
     -o {1}"

[[VALIDATORS]]
NAME="ec2-52-15-193-153.us-east-2.compute.amazonaws.com"
HOME_DOMAIN="botika.ai"
PUBLIC_KEY="GDU5D5DTNBMGTN6XPMWX5PAT5MBM5BDG7ZW2GNLOEZHRH5L6YAZ6FETX"
ADDRESS="ec2-52-15-193-143.us-east-2.compute.amazonaws.com"
HISTORY="curl -sf ec2-52-15-193-143.us-east-2.compute.amazonaws.com/vs/{0}
    -o {1}"
```

## 4.2 BTK Layer

Per l'implementazione delle API precedentemente descritte è stato utilizzato Swagger. Questo tool ha offerto la possibilità di generare automaticamente la struttura del server API scritto in Go partendo da un file *.yaml* . Una volta svolto questo passaggio è stato necessario implementare solamente gli handler di ogni singolo endpoint definito durante la fase di progettazione. Qui di seguito, come esempio, viene riportato il codice della funzione di creazione di un asset. Questa funzione è richiamata nell'handler di gestione dell'API **POST/assets/** e utilizza spesso funzioni descritte all'interno delle classi *AssetService* ed *AccountService*.

```
1  func CreateAssetFn(accountService *services.AccountService, assetService *
       services.AssetService) func(args assets.CreateAssetParams) middleware.
       Responder {
2    if accountService == nil || assetService == nil {
3      return nil
4    }
5    return func(args assets.CreateAssetParams) middleware.Responder {
6
7      assetCode := args.Body.AssetCode
8      amount := args.Body.Amount
9      distributorPublicKey := args.Body.PublicKey
10
11     err := assetService.ValidateAssetCode(assetCode)
12     if err != nil {
13       return assets.NewCreateAssetUnprocessableEntity()
14     }
15
16     _amount, err := strconv.Atoi(amount)
17     if err != nil || _amount <= 0 || distributorPublicKey == "" {
18       return assets.NewCreateAssetUnprocessableEntity()
19     }
20
21     distributorAccount, err := accountService.FindAccountOnDatabase(
       distributorPublicKey)
22     initialBalance := accountService.ConfigService.Config.StellarConfig.
       InitialBalance
23     _ops := []txnbuild.Operation{}
```

```
24
25    if err != nil || distributorAccount == nil {
26      return assets.NewCreateAssetInternalServerError()
27    }
28
29    // Check if distributorAccount has a related issuerAccount. If not, we
      need to create one
30    var issuerAccount models.Account
31    if distributorAccount.HasIssuer {
32      issuerAccount = *distributorAccount.RelatedAccount
33    } else {
34      // Create issuer account related to this distributor account
35      kp, err := accountService.GenerateKeyPairs()
36      if err != nil {
37        return assets.NewCreateAssetInternalServerError()
38      }
39
40      // Generate the 'createAccount' operations
41      createAccountOps, err := accountService.HorizonService.
      CreateAccountsOps([]*keypair.Full{kp}, initialBalance)
42      if err != nil {
43        return assets.NewCreateAssetInternalServerError()
44      }
45
46      // Append create account ops
47      _ops = append(_ops, createAccountOps...)
48
49      // Create account in db
50      _issuerAccount, err := accountService.CreateIssuerInDatabase(kp)
51      if err != nil {
52        return assets.NewCreateAssetInternalServerError()
53      }
54      issuerAccount = *_issuerAccount
55
56      err = accountService.UpdateAccountWithIssuerRelation(&issuerAccount,
      distributorAccount)
57      if err != nil {
58        return assets.NewCreateAssetInternalServerError()
59      }
60    }
61
```

```
62      // Add Master Account to signers
63      masterKp, err := accountService.HorizonService.MasterKp()
64      if err != nil {
65        return assets.NewCreateAssetInternalServerError()
66      }
67      signers := []*keypair.Full{masterKp}
68
69      // Generate internal asset model
70      internalAsset := &models.InternalAsset{
71        Issuer: &issuerAccount,
72        Code:   assetCode,
73      }
74
75      //generate horizon asset model
76      asset, err := assetService.Asset(internalAsset)
77      if err != nil {
78        return assets.NewCreateAssetInternalServerError()
79      }
80
81      //generate the 'changeTrustline' operations
82      //used to create a trustline between issuer and distributor accounts
83      createTrustlineOps, err :=
84        assetService.HorizonService.CreateTrustlineOps([]string{
85          distributorPublicKey,
86        }, asset)
87      if err != nil {
88        return assets.NewCreateAssetInternalServerError()
89      }
90
91      distributorKp := keypair.MustParseFull(distributorAccount.SecretKey)
92      // sign as distributor for CreateTrustlineOps
93      signers = append(signers, distributorKp)
94
95      //generate the 'createFund' operations
96      //used to give tokens to the distributor account
97      createFundOps, err :=
98        assetService.HorizonService.CreateFundOps(&issuerAccount.PublicKey,
        []string{
99          distributorAccount.PublicKey,
100       }, asset, amount)
101     if err != nil {
```

```
102       return assets.NewCreateAssetInternalServerError()
103     }
104     // sign as issuer for FundOps
105     issuerKp := keypair.MustParseFull(issuerAccount.SecretKey)
106     signers = append(signers, issuerKp)
107
108     //append operations
109     _ops = append(_ops, createTrustlineOps...)
110     _ops = append(_ops, createFundOps...)
111
112     //create transaction with (optionally) multi-sign
113     _tx, err := assetService.HorizonService.CreateTransaction(nil, _ops,
        signers...)
114     if err != nil {
115       return assets.NewCreateAssetInternalServerError()
116     }
117
118     //submit transaction
119     txRes, err := assetService.HorizonService.SubmitTx(_tx)
120     if err != nil {
121       return assets.NewCreateAssetInternalServerError()
122     }
123
124     assetName, err := assetService.GetAssetName(internalAsset)
125     if err != nil {
126       return assets.NewCreateAssetInternalServerError()
127     }
128
129     return assets.NewCreateAssetCreated().WithPayload(&dto.
        CreateAssetResponseDTO{
130       Amount:                amount,
131       AssetCode:             assetCode,
132       AssetName:             *assetName,
133       DistributorAccount:    args.Body.PublicKey,
134       IssuerAccount:         issuerAccount.PublicKey,
135       TransactionSuccessful: &txRes.Successful,
136     },
137     )
138   }
139 }
```

Dopo aver salvato gli argomenti in ingresso della chiamata in variabili viene controllata la validità di questi. Ad esempio viene controllato se esiste un account con la chiave pubblica sul database e se la somma di token che si intende generare sia maggiore di zero. Terminato il controllo di tutti gli argomenti presi in input si controlla se colui che intende generare l'asset possiede già un account di tipo issuer. Nel caso non lo avesse già ne verrà creato uno apposito prima di poter procedere. Successivamente viene creata un'operazione di trustline riguardo l'asset in oggetto fra l'account issuer e l'account distributore. Questa operazione è necessaria per permettere lo scambio di token di quello specifico asset tra i due account. Grazie alla trustline appena generata l'account distributore accetterà di ricevere pagamenti che hanno come valuta l'asset appena creato. Come si può notare a riga 100 per generare effettivamente token viene poi effettuata un'operazione di pagamento dove vengono indicate le chiavi pubbliche dei due account, l'asset da creare e la quantità di questo. Tutto il procedimento appena descritto è mostrato anche nel diagramma di sequenza in figura 4.1.

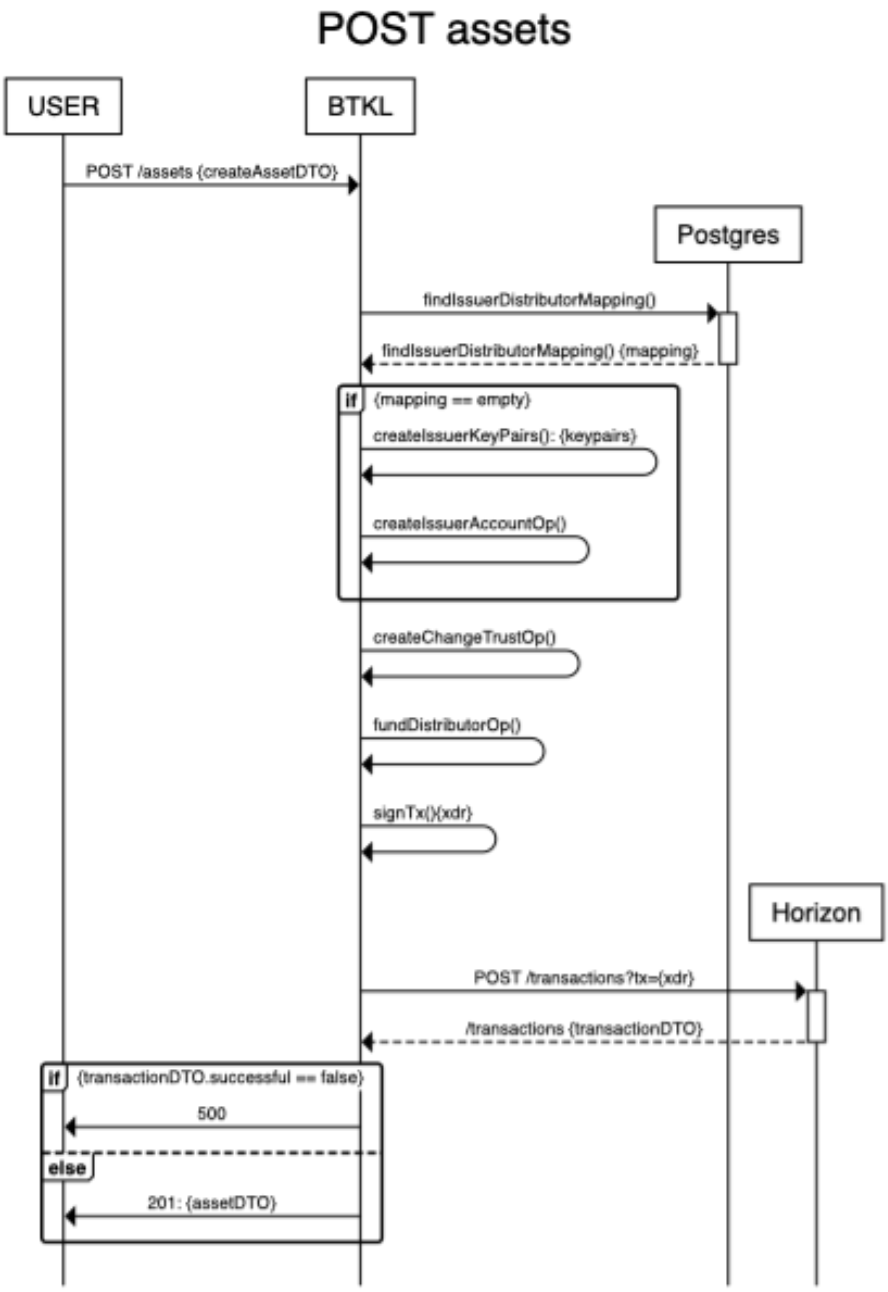


Figura 4.1: Diagramma di sequenza: POST/assets/

# Capitolo 5

# Sviluppi futuri

Lo scopo principale di questo progetto di tesi è stato quello di realizzare un sistema di pagamenti basato su un sistema distribuito. Allo stato attuale la piattaforma realizzata può solamente effettuare scambi che hanno come oggetto utility token creati sulla rete.
Un token di utility rappresenta l'accesso a futuri prodotti e servizi di un'azienda. Dietro un utility token non c'è alcun prodotto o servizio tangibile. Questi token quindi non sono progettati per essere un investimento finanziario, ma ottengono il loro valore dalla piattaforma stessa.
A differenza di quasi tutti gli altri sistemi blockchain, Stellar può connettersi a endpoint del mondo reale, così gli utilizzatori del sistema possono trasformare le loro rappresentazioni digitali di denaro (token) in qualcosa che possono effettivamente spendere. Questi endpoint prendono il nome di *anchors*. Un'ancora (anchor) è un'entità fidata sulla rete Stellar che è autorizzata a trattenere i depositi dell'utente ed emettere credito per tali depositi. Le anchors fungono da ponte tra le valute reali e la rete Stellar emettendo crediti agli utenti equivalenti al valore effettivo della valuta sotto forma di assets. Gli utenti possono quindi utilizzare questo credito per avviare le transazioni sulle reti Stellar. Le anchors solitamente tendono ad essere organizzazioni come istituzioni finanziarie, società di rimesse e banche centrali.
Stellar sta aprendo la strada verso un nuovo standard di pagamento globale e le anchors hanno l'opportunità di diventare parte integrante di questo sforzo. Come collegamento critico tra la rete Stellar e il sistema bancario tradizionale nei rispettivi paesi, le ancore sono posizionate per sfruttare una varietà

di modelli di business e strategie di monetizzazione, comprese le commissioni di deposito/prelievo, spread FX e commissioni di transazione. Le anchors di successo offrono servizi di ancoraggio come parte di un più ampio portafoglio di prodotti e servizi che sfruttano l'efficienza di Stellar per fornire servizi finanziari nuovi e convenienti, come ad esempio le offerte di *Banking-as-a-Service* basate su API, i pagamenti e le fatturazioni B2B transfrontaliere, i bonifici, i pagamenti P2P, il libro paga internazionale o i pagamenti su due lati del mercato.
Oltre a supportare l'emissione e il movimento di asset, la rete Stellar funge anche da *exchange* distribuito e decentralizzato che consente di scambiare e convertire asset sulla rete. Il libro mastro Stellar memorizza sia i saldi detenuti dagli account sia gli ordini che gli utenti effettuano per acquistare o vendere beni. Un utente può creare ordini per acquistare o vendere beni utilizzando operazioni come *Manage Buy Offer* o *Manage Sell Offer*. Per avviare un ordine, l'account deve detenere l'asset che desidera utilizzare per acquistare o scambiare l'asset desiderato. L'account deve inoltre anche fidarsi dell'issuer della risorsa che sta cercando di acquistare.
Supponendo di tenere nel proprio saldo alcuni token "**A**" e di voler acquistare qualcosa da un negozio che accetta solo token di tipo "**B**", in Stellar si può creare un pagamento che converte automaticamente i token "**A**" in "**B**". Passando attraverso l'*orderbook* **A/B** la rete Stellar converte i tuoi token alla migliore tariffa disponibile.
È possibile anche creare percorsi più complicati di conversione delle risorse. Immaginando che l'orderbook **A/B** abbia uno spread molto ampio o sia inesistente, si potrebbe ottenere una tariffa migliore se prima venissero scambiato i token "**A**" con token di tipo "**C**" e poi vengano nuovamente scambiati questi ultimi. Così si otterrebbero token "**B**". Quindi un potenziale percorso sarebbe: "**A**" ->"**C**" ->"**B**". Questo percorso condurrà attraverso l'orderbook di **A/C** e quindi l'orderbook di **C/B**.
Questi percorsi di conversione delle risorse possono contenere fino a 6 orderbook differenti, mantenendo l'intero pagamento atomico. L'intero processo descritto avrà esito o positivo o negativo. Il mittente del pagamento non sarà quindi mai lasciato in possesso di un bene da lui indesiderato.
Poiché i pagamenti *cross-asset* sono molto semplici grazie a Stellar, gli utenti

possono mantenere il loro denaro in qualsiasi tipologia di asset. Queste funzionalità di Stellar creano un sistema aperto e molto flessibile.
Il sistema progettato è certamente aperto ad interessanti sviluppi, che possono apportare migliorie sia per gli utilizzatori che per gli attori della rete.
Per i primi potrebbero essere implementate le seguenti funzionalità:

- implementare app mobile da cui poter effettuare agilmente le chiamate API descritte in precedenza.
- implementare, grazie all'utilizzo di anchors Stellar, la gestione di pagamenti P2P con qualsiasi tipo di valuta comprese valute reali.
- introdurre la possibilità di effettuare trading sulla rete.

Inoltre, per rendere l'intera rete più sicura e maggiormente distribuita, grazie alla scalabilità intrinseca della rete, si potrebbe aumentare il numero di attori con successiva integrazione di nuovi nodi Stellar.

# Capitolo 6

# Conclusioni

Il lavoro svolto in questo elaborato di tesi ha fornito un' analisi dettagliata riguardo la blockchain di Stellar ed allo stesso tempo ha permesso di valutarne la fattibilità di applicazione in un contesto finanziario. L'industria finanziaria è stata una delle prime a cogliere le opportunità derivanti dalle tecnologie Blockchain e, in generale, dalle Distributed Ledger Technologies. Le ragioni di questo interesse non andrebbero però ricercate soltanto nella necessità di partecipare ad un mercato delle criptovalute ormai in forte ascesa.
Quello del Finance è infatti un comparto per diversi aspetti fortemente vincolato ad architetture legacy, molte delle quali frutto di implementazioni e upgrade sviluppati nell'arco di decenni, ma difficilmente rimpiazzabili per questioni inerenti l'efficienza, l'entità dell'investimento necessario ad una migrazione verso altre piattaforme, la sicurezza e la mancanza di alternative sufficientemente affidabili.
Dal punto di vista dell'ottimizzazione dei costi e della massimizzazione del *Return Of Investment*, l'impiego delle tecnologie basate sulla Blockchain è in grado di determinare benefici elevati soprattutto quando mirato alla condivisione. Un utilizzo sistematico e strutturale delle DLT si potrebbe tradurre, ad esempio, in una riduzione notevole dei costi operativi per gli istituti finanziari. Le ragioni di questo fenomeno andrebbero ricercate nel fatto che una Blockchain condivisa, quindi basata su specifiche valide per tutti gli operatori del settore e recepite da una buona parte di essi, richiederebbe un impegno economico inferiore rispetto ad un sistema in cui ciascun istituto utilizzi la propria piattaforma per la gestione delle transazioni.

A tal proposito sarebbe sufficiente citare alcuni fattori fino ad oggi determinanti nel definire i margini di profitto, come per esempio le commissioni per i soggetti garanti, i costi per i trasferimenti monetari e le commissioni per le operazioni di intermediazione. Tutte componenti che verrebbero rimosse o fortemente ridimensionate nel caso in cui si disponesse di un sistema P2P, distribuito, interoperabile e, come anticipato, condiviso.

L'implementazione della rete, in questo progetto di tesi, è avvenuta con successo ed il modello realizzato consente di svolgere tutte le operazioni descritte nei capitoli precedenti.
I tempi di sviluppo sono stati dell'ordine dei 4 mesi uomo.
Lo sviluppo della piattaforma è stato articolato in 3 fasi:

- **documentazione e definizione funzionalità**: dopo aver studiato la documentazione ufficiale di Stellar si è cercato di capire in che modo esporre ed implementare le funzionalità richieste.
- **configurazione nodi Stellar e test dimensionamento hardware**: per poter definire le caratteristiche hardware necessarie allo sviluppo della rete è stato opportuno configurare nodi Stellar. Terminata questa fase sono stati effettuati stress test nei quali è stato monitorato il carico di utilizzo per ognuna delle componenti.
- **definizione ed implementazione di API**: nell' ultima fase del progetto sono state tradotte le funzionalità della piattaforma in API. Queste sono state descritte in un file .yaml e successivamente convertite in un webservice tramite Swagger. Grazie a questa operazione si è dovuto implementare solamente l'handler dei vari endpoint.

# Bibliografia


[1] *"Distributed Ledger Technology Blockchain as Middleware"* - Andrea Omicini, Giovanni Ciatto

[2] *"The Problem of Consensus in Distributed Systems"* - Andrea Omicini

[3] *BlockChain4Innovation* -https://www.blockchain4innovation.it/esperti/blockchain-perche-e-cosi-importante/

[4] *How does a blockchain work? Simply explained-* https://www.youtube.com/watch?v=SSo_EIwHSd4

[5] *Stellar* - https://www.stellar.org

[6] *Lumenauts* - https://www.lumenauts.com

[7] *SCP* - https://medium.com/interstellar/understanding-the-stellar-consensus-protocol-423409aad32e

[8] *Simplified SCP* - https://www.scs.stanford.edu/ dm/blog/simplified-scp.html

[9] *Stellar Documentation* - https://developers.stellar.org/docs/

[10] *Stellar Quorum* - https://www.stellar.org/developers-blog/why-quorums-matter-and-how-stellar-approaches-them

[11] *Stellar Blockchain* - https://www.leewayhertz.com/what-is-stellar-blockchain/

[12] *Blockchain & Finance* - https://www.modis.com/it-it/insights/articoli/blockchain-finance/

[13] *White paper SCP* - https://assets.website-files.com/5deac75ecad2173c2ccccbc7/5df2560fba2fb0526f0ed55f_stellar-consensus-protocol.pdf