UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Tesi di Laurea Triennale

SVILUPPO DI UN SISTEMA DI RACCOMANDAZIONE SULLA BLOCKCHAIN EOS.IO

Relatori:
Prof.ssa *Laura Emilia Ricci*
Dott. *Andrea De Salve*

Candidato:
Giuseppe Sacco

ANNO ACCADEMICO 2020-2021

Abstract

Il tirocinio consiste nello studio, progettazione, implementazione e valutazione di un sistema di recensione sviluppato sulla blockchain EOS.IO che fa uso di token. I token vengono assegnati ad un utente che effettua una recensione in base all'esperienza acquisita. Per stimolare gli utenti ad usufruire del servizio offerto dal sistema, i token possono essere utilizzati come risorsa economica per effettuare pagamenti. L'esperienza dell'utente nell'ambito del sistema di recensione è determinata dalle skills acquisite.

Indice

Ringraziamenti

Ringrazio prima di tutto la Professoressa Laura Ricci e il Dott. Paolo Mori, per i suggerimenti, le preziose indicazioni e il sostengo che mi hanno dimostrato in questi mesi riuscendo ad essere presenti nonostante i vari impegni.

Un immenso ringraziamento al Dott. Andrea De Salve per essere sempre stato presente a qualsiasi orario, per chiarimenti, confronti e preziosi consigli.

Grazie anche al Dott. Andrea Lisi per la sua disponibilità e per i suoi indispensabili suggerimenti.

Mamma, papà, grazie per avermi permesso di fare le mie scelte, di sbagliare e di raggiungere i miei traguardi. Puoi prendere o lasciare, puoi volere, puoi lottare o fermarti e rinunciare.

Grazie alla mia famiglia, alle persone che anche senza saperlo mi hanno mostrato la via e sono state per me una fonte di ispirazione: mio nonno, Giuseppe, Rocco. Infondo, non impari dalla pioggia se non ti ci infradici.

Grazie Marco, Sunday e Marianna per essere il fratello e le sorelle che non ho mai avuto. Grazie ai miei amici di Senise. E che metti che ho un sacco di sbatti, ci si sbattono un sacco a distrarmi.

Grazie SF32 e GBP14 per essere stati casa quando la mia era lontana. A tutti i miei colleghi universitari di Pisa. Non so se chiamare casa questo posto, o i chilometri che ho fatto per arrivarci.

Grazie a tutte le persone che hanno fatto parte della mia vita e a tutte quelle che ne fanno ancora parte; perchè la vita è un aereoporto, tutti vanno e vengono.

Capitolo 1

Introduzione

La diffusione delle tecnologie informatiche e la crescente diffusione dei contenuti digitali offrono agli utenti la possibilità di utilizzare i media digitali in ogni occasione. La personalizzazione dell'offerta è diventata un fattore cruciale per molte aziende orientate alla vendita online. Grande interesse è stato rivolto all'analisi e alla modellizzazione dei gusti individuali delle persone, con l'obiettivo di suggerire nuovi prodotti di loro possibile gradimento.

Questo meccanismo è alla base dei cosiddetti sistemi di recensione, ormai adottati da diverse piattaforme in svariati settori. Ad esempio, Google My Business per le aziende che vogliono promuovere la propria attività, oppure Facebook Places per la recensione dei luoghi, dove è possibile inserire foto e molto altro. Amazon e Ebay utilizzano un sistema di recensione per i prodotti venduti e per raccogliere gli apprezzamenti o meno degli utenti. Altre piattaforme come Booking o Trivago con caratteristiche simili, permettono di recensire l'alloggio che l'utente ha prenotato e successivamente soggiornato. Simile alle ultime ma con uno scopo più ampio, Tripadvisor permette di recensire anche i ristoranti.

I sistemi di recensione (RS), talvolta noti anche come sistemi di raccomandazione, collezionano le recensioni pubblicate dagli utenti al fine di valutare il servizio offerto da una data attività.

Il calcolo del rating determina la reputazione dell'attività digitale ed è rilevante sia per il proprietario della stessa, sia per gli utenti che vogliono

usufruire del servizio offerto [47].

I sistemi di recensione più diffusi sono basati su architetture centralizzate gestite da grandi multinazionali. Il fornitore dei servizi di queste applicazioni controlla e memorizza tutti i dati pubblicati dagli utenti utilizzando algoritmi di rating che non sempre vengono resi pubblici. Infatti, uno dei principali problemi dei RS attualmente in circolazione, è la mancata fiducia verso i meccanismi utilizzati all'interno di queste applicazioni.

I RS centralizzati sono controllati da una autorità centrale che gestisce e verifica tutte le informazioni. L'autorità centrale può decidere in maniera arbitraria quali recensioni considerare e determinare il successo o l'insuccesso di determinati item. I dati delle recensioni possono essere resi disponibili, a discrezione dell'autorità centrale che può decidere quali e quanti dati rendere visibili. L'autorità centrale ha la possibilità di influenzare il processo di ranking e decidere di rimuovere le opinioni degli utenti.

La decentralizzazione è una soluzione promettente per tali problemi [42]. Può essere vista come una valida alternativa alle società centralizzate, quali Amazon, Google, Apple e così via. Quello che la decentralizzazione rappresenta è un metodo di comunicazione in cui nessun ente centralizzato ha il controllo sui contenuti condivisi tra gli utenti [51].

La tecnologia blockchain è la concretizzazione dell'idea di decentralizzazione. L'avvento di Bitcoin [48] ha portato ad uno studio più approfondito al fine di comprendere le potenzialità che avrebbe potuto offrire una simile tecnologia.

L'obiettivo era quello di permettere la costruzione di sistemi decentralizzati che offrissero dei servizi alla comunità. Non sarebbe bastato creare una semplice criptovaluta, per raggiungere lo scopo serviva costruire una vera e propria piattaforma che permettesse lo sviluppo di ecosistemi che utilizzassero la tecnologia blockchain. Uno tra i tanti obiettivi raggiunti e/o in sviluppo è EOS.IO [27]. Questa recente blockchain, permette la scrittura di smart contract e il conseguente sviluppo di applicazioni decentralizzate. L'interazione con il software EOS.IO implica dei costi, motivo per cui possiede la propria criptovaluta utilizzata dagli utenti per eseguire le transazioni e non solo.

1.1 Motivazioni e contributi

Una recente tecnica utilizzata per risolvere i problemi introdotti dall'autorità centrale à quella di implementare i rating system sfruttando le blockchain. Queste applicazioni esistono già sulla blockchain Ethereum [30], con svariati vantaggi e svantaggi. Le blockchain introducono dei costi per l'esecuzione delle transazioni, così anche Ethereum introduce il concetto del gas, per pesare il costo delle operazioni che modificano lo stato del sistema, ad ogni unità di gas è associato un valore in *ether*, la criptovaluta nativa di *Ethereum*. Gli svantaggi che questa blockchain porta con se, sono problemi di performance e scalabilità. Infatti, i tempi tecnici che Ethereum impiega per l'esecuzione di 15 transazioni è di circa un secondo, con un tempo di conferma pari a un minuto [32].

Recentemente, la piattaforma EOS.IO e' stata proposta come blockchain alternativa e promette di risolvere il problema dei costi ammortizzandoli nel tempo. Inoltre, introduce delle prestazioni migliori rispetto alle piattaforme più popolari come Ethereum stessa e Bitcoin.

L'obiettivo del tirocinio è implementare il rating system framework in *EOS.IO*, precedentemente realizzato per Ethereum. Lo scopo è confrontare le due implementazioni ed evidenziare quelle che sono le differenze tra le due blockchain, valutando nel contempo i pregi e i difetti che esse presentano.

Per arrivare al risultato finale, è stato necessario uno studio approfondito della blockchain presa in analisi, comparandola con quelle che sono le caratteristiche già note di Ethereum.

Il rating system è basato su un sistema di tokenizzazione. Tokenizzare il sistema, significa creare un'economia interna introducendo il concetto di token al fine di migliorare l'esperienza utente. Il token ha la funzione di risorsa digitale come mezzo di accesso a beni o a servizi.

Lo sviluppo, il deploy e l'utilizzo della così detta DApp (Decentralized Application) comporta anche dei costi. Lo scopo di questo studio è anche evidenziare, se sono presenti, i pregi di EOS.IO in termini di costi di gestione combinati con le performance offerte. Questi, permettono allo stesso tempo di mettere in risalto gli svantaggi di Ethereum, incentivando all'utilizzo della blockchain presa in analisi. All'interno della blockchain EOS, come in Ethe-

reum, ci sono dei costi da sostenere anche nel momento in cui un utente deve inserire una recensione. A tal proposito, l'incentivo ad utilizzare il sistema è dato dall'impiego dei token: attraverso la creazione di un token, un proprietario di un'attività può utilizzarli per ricopensare il cliente della recensione inserita.

I token potranno essere utilizzati come buoni sconto presso l'attività o il servizio che li ha rilasciati. Inoltre, il framework integra delle operazioni per il calcolo del rating tramite media oppure media pesata in base all'esperienza dei recensori. Attraverso quest'ultima, un utente può visualizzare il punteggio assegnato ad un servizio che attribuisce una maggiore rilevanza allo score pubblicato da un utente esperto.

Per raggiungere lo scopo, il sistema implementato è stato deployato e testato sulla testnet *Jungle3*.

Al fine di dimostrare la fattibilità della strategia di tokenizzazione operata e per comprendere il costo delle varie operazioni, abbiamo sviluppato un prototipo del sistema conducendo un'analisi critica per valutare le sue performance.

I risultati ottenuti sono stati dunque paragonati all'implementazione fatta in Ethereum. Ne è emerso che, il sistema realizzato con EOS.IO offre varie potenzialità. Inoltre, costi di gestione e costi per l'esecuzione delle transazioni sono più convenienti per gli utenti che vogliono interagire con il sistema di recensione.

1.2 Road map

Il documento è organizzato come segue: il capitolo 2 introduce il lettore ai concetti fondamentali legati al mondo delle blockchain, mettendo in risalto i processi per la validazione delle transazioni e fornisce una panoramica sugli smart contract.

Il terzo capitolo è un'introduzione a EOS, mettendo in evidenza tutti gli aspetti, che secondo alcuni studi, devono essere analizzati per valutare una blockchain. Durante la descrizione di tutti i punti focali che distinguono le

blockchain, vengono messe a paragone EOS ed Ethereum, evidenziando quelli che possono essere i pro o i contro sotto svariati punti di vista.

Il capitolo 4 presenta l'architettura del sistema di recensione basato sui token, evidenziando come i moduli della struttura sono relazionati tra loro, e come collaborano per raggiungere il corretto funzionamento delle specifiche del sistema. Attraverso la descrizione delle operazioni che gli utenti possono effettuare, viene mostrato il funzionamento e come avviene l'interazione con il sistema.

Il capitolo 5 è dedicato interamente alle scelte implementative adottate per la realizzazione del framework, che si distinguono interamente dall'applicazione sviluppata per Ethereum a causa dei diversi metodi di programmazione utilizzati. Inoltre, vi è una descrizione degli strumenti utilizzati nella fase di sviluppo e quelle che sono le differenze strutturali prese in fase di implementazione.

Il capitolo 6 si focalizza sulle valutazioni delle performance del framework in relazione a quelli che sono i consumi delle risorse fatte da ogni account in fase di interazione con la blockchain. Durante tutto il capitolo vengono evidenziati i costi anche in termini di EURO. Inoltre, vengono riportati i risultati dei test effettuati al sistema. Successivamente, viene riportato un calcolo approssimativo di quelli che sono i costi che un utente deve sostenere per mantenere il suo locale sulla blockchain. Infine, verranno messi a confronto le implementazioni di EOS.IO e di Ethereum, evidenziando quelle che sono le discrepanze di costi delle varie possibili operazioni.

Il capitolo 7 presenta una considerazione finale sul lavoro proposto nel tirocinio che punta a un'analisi delle novità introdotte. Infine, viene effettuata una breve panoramica sui possibili miglioramenti futuri che potrebbero essere apportati.

Capitolo 2

Fondamenti

In questo capitolo introduciamo il lettore alle principali nozioni di base relative alle piattaforme blockchain, all'algoritmo di consenso, e alla definizione di smart contract. Infine, vedremo come questi concetti sono stati sviluppati all'interno della piattaforma EOS.IO.

2.1 Blockchain

La blockchain è una struttura dati, condivisa e immutabile, composta da un insieme di blocchi contenenti transazioni [39] (figura 2.1).

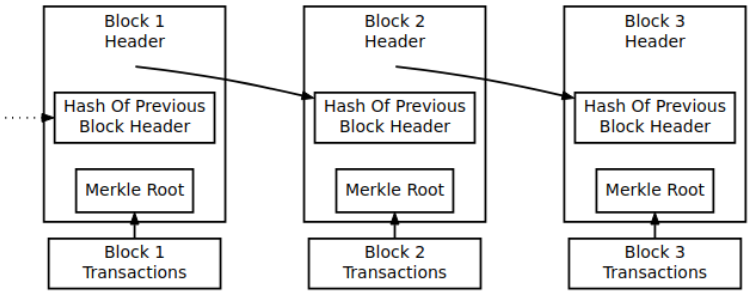


Figura 2.1: Semplificazione della blockchain di Bitcoin [44]

Un blocco è una struttura dati che consente di memorizzare un file elenco delle transazioni. Le transazioni vengono create e scambiate dai nodi della rete che modificano lo stato della blockchain. In quanto tali, le transazioni possono scambiare denaro, ma non sono limitati alle sole transazioni finanziarie.

Le nuove generazioni di blockchain, infatti, consentono di eseguire del codice all'interno dei cosiddetti smart contract [53].

La tecnologia blockchain consente di creare un database sicuro basandosi su un protocollo di consenso distribuito che coinvolge i nodi della rete.
In sostanza, potremmo dire che la blockchain non è altro che la combinazione di tre diverse tecnologie:

- peer-to-peer networking: paradigma che si contrappone alla classica architettura client-server. Indica un modello di architettura logica di rete informatica in cui i nodi non sono gerarchizzati sotto forma di client o server fissi, ma piuttosto risultano essere equivalenti o definiti paritari.

- crittografia asimmetrica: conosciuta anche come crittografia a chiave pubblica. Viene definita asimmetrica poiché viene utilizzata una chiave pubblica per cifrare e una chiave privata per decifrare il messaggio. Ciò vuol dire che, essendo una chiave pubblica, chiunque può cifrare un messaggio; ma soltanto chi possiede la chiave privata potrà decifrarlo [25]. La robustezza di un sistema di crittografia a chiave pubblica si basa sulla difficoltà di determinare la chiave privata corrispondente alla chiave pubblica.

- hashing crittografico: si tratta di un insieme di algoritmi matematici con determinate proprietà che garantiscono la sicurezza. Una funzione hash è una funzione non invertibile che mappa dati di lunghezza arbitraria in una stringa binaria di dimensione fissa chiamata valore di hash.

La combinazione di questi elementi consente lo sviluppo e il mantenimento di un database decentralizzato. Nella blockchain ogni blocco contiene l'hash del blocco precedente, il quale serve per garantire integrità, replicabilità e determinismo [32]. Tutte le transazioni vengono raccolte nel *ledger* (libro mastro).

Volendo far spazio ad altre terminologie nell'ambito delle blockchain, possiamo citare il *genesis block* (o blocco di genesi). Il blocco di genesi è il primo blocco di ogni blockchain. Questo tipo di blocco viene utilizzato per avviare una criptovaluta. Storicamente fu Satoshi Nakamoto ad inserire il primo blocco

nella catena di Bitcoin. Qualsiasi blockchain avrà un blocco con caratteristiche simili che avvia la propria catena.

Le transazioni sono il modo di interagire con la blockchain in forma di pacchetti dati che contengono le informazioni. In generale, nel momento in cui una transazione viene effettuata, essa viene propagata ai nodi connessi che validano e diffondono la transazione finché non raggiunge tutti i nodi restanti della rete.

Ogni nodo possiede una copia privata della blockchain. La validazione di un blocco, e delle transazioni contenute in esso, è verificata con il consenso dai partecipanti al sistema mediante un algoritmo di consenso usando la crittografia.

Come è normale che sia, ogni blockchain è strutturata per far fronte a delle determinate esigenze, fornire determinate prestazioni soprattutto per questione di necessità di utilizzo. A tal proposito esistono blockchain definite *permissioned* e altre *permissionless*. Come vedremo meglio nella sezione 3.4, esistono diverse configurazioni di blockchain in base alle autorizzazioni richieste per poter accedere alla rete, eseguire delle transazioni o partecipare all'algoritmo di consenso.

2.2 Algoritmi di consenso

È bene distinguere un protocollo di consenso da un algoritmo di consenso: il protocollo di consenso è l'insieme delle regole primarie di una blockchain; un algoritmo di consenso è il meccanismo attraverso cui queste regole vengono fatte rispettare.

Nel contesto delle criptovalute, gli algoritmi di consenso costituiscono un elemento cruciale per ogni network blockchain, in quanto hanno il compito di mantenere l'integrità e la sicurezza di questi sistemi distribuiti.

Un algoritmo di consenso viene utilizzato da un insieme di entità non fidate per concordare in maniera consistente la corretta sequenza delle azioni ed il loro risultato. Serve per concordare l'ordine dei blocchi/transazioni che sono valide [22].

Essi garantiscono che le regole del protocollo vengano seguite e che tutte le transazioni avvengano correttamente [3].

Ad esempio, l'algoritmo di consenso di una blockchain è ciò che determina la validità delle transazioni e dei blocchi. Quindi Bitcoin, Ethereum e EOS.IO sono alcuni dei protocolli, mentre la Proof of Work, la Proof of Stake e la Delegated Proof of Stake sono alcuni degli algoritmi di consenso.

La scelta dell'algoritmo di consenso evidenzia i vari pregi e difetti del protocollo di consenso. L'obiettivo rimane comunque quello di bilanciare la sicurezza e sfruttare funzionalità e scalabilità del sistema.

Nella maggior parte dei protocolli di consenso, una transazione è affiancata da una tassa, la quale diventa un incentivo per i nodi che eseguono la transazione a rimanere onesti.

La tassa pagata diventa una ricompensa per i nodi che validano la transazione attraverso lo svolgimento di una computazione, in genere complessa [32]. Nel momento in cui una transazione viene validata, essa entra a far parte di un blocco, il quale viene aggiunto alla blockchain come ultimo componente della catena. Questo processo viene chiamato *mining* in algoritmi di consenso come la proof of work.

Figura 2.2: Differenza tra PoW, PoS e DPoS[45]

Chiariamo ulteriormente la differenza tra algoritmo e protocollo di consenso. Il protocollo utilizzato definisce il modo in cui i nodi interagiscono, come i dati devono essere trasmessi tra loro, quali sono i requisiti per convalidare un blocco con successo, performance, autenticazione, integrità, non ripudio e tolleranza ai guasti. D'altro canto, l'algoritmo di consenso ha il compito di verificare i bilanci e le firme, il quorum, confermare le transazioni ed eseguire effettivamente la convalida di blocchi.

Quindi, protocollo di consenso e algoritmo di consenso combinati tra loro non sono altro che il cuore della blockchain e ne delineano le caratteristiche che andremo ad analizzare.

2.3 Smart contract

Uno smart contract è un programma che risiede sulla blockchain, la sua esecuzione è condizionata dal protocollo di consenso. Uno smart contract può codificare qualsiasi insieme di regole rappresentato nel suo linguaggio di programmazione. Ad esempio, un contratto può eseguire trasferimenti di una criptomoneta quando si verificano determinati eventi. Di conseguenza, i contratti intelligenti possono implementare un'ampia gamma di applicazioni, inclusi strumenti finanziari o applicazioni autonome di governance [37].

Il significato letterale di smart contract è "contratti intelligenti". In concreto, la potenzialità degli smart contract è consentire l'esecuzione di transazioni e accordi affidabili tra parti diverse senza la necessità di un'autorità centrale, un sistema legale o un meccanismo di applicazione esterno.

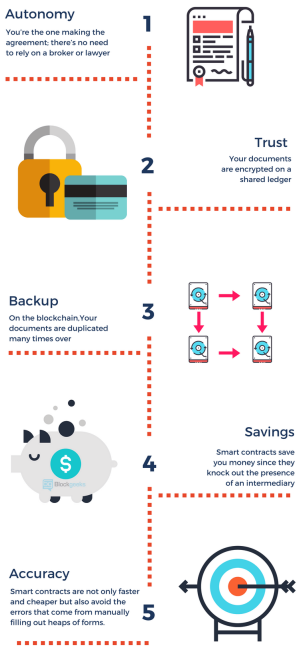


Figura 2.3: Benefici degli smart contract [34]

Figura 2.4: Flusso di un contratto tradizionale [17]

Inizialmente, gli smart contract erano intesi come l'equivalenza in digitale di un contratto cartaceo, in grado di definire regole e penali di un accordo: un contratto autoeseguente in cui i termini dell'accordo tra acquirente e venditore vengono scritti direttamente in righe di codice, facendo uso delle istruzioni condizionali "if-then-else", le quali gestiscono i pagamenti al verificarsi di determinate condizioni.

Ad oggi, invece, per smart contract si intende l'esecuzione di qualsiasi generica computazione che entra a far parte della blockchain.

Più in generale, gli smart contract aiutano le persone a scambiare denaro, trasferire proprietà e qualsiasi altra cosa di valore in modo trasparente e

senza ricorrere ai servizi di un intermediario. Il codice e gli accordi in esso contenuti vengono distribuiti ed eseguiti sulla blockchain come parte della convalida di una transazione. Il codice controlla l'esecuzione e le transazioni sono tracciabili e irreversibili. Uno smart contract è un programma, scritto in un particolare linguaggio di programmazione, che viene eseguito come programmato, senza possibilità di censura o manipolazione. Il codice prodotto è legge e ha l'obiettivo di facilitare le operazioni tra persone ed istituzioni.

	Ethereum	**Fabric**	**Corda**	**Stellar**	**Rootstock**	**EOS**
Execution environment	EVM	Docker	JVM	Docker	VM	WebAssembly
Language	Solidity, Serpent, LLL, Mutan	Java, Golang	Java, Kotlin	Python, JavaScript, Golang and PHP, etc.	Solidity	C++
Turing Completeness	Turing complete	Turing complete	Turing incomplete	Turing incomplete	Turing complete	Turing complete
Data model	Account-based	Key–value pair	Transaction-based	Account-based	Account-based	Account-based
Consensus	PoW	PBFT	Raft	Stellar Consensus Protocol (SCP)	PoW	BFT-DPOS
Permission	Public	Private	Private	Consortium	Public	Public
Application	General	General	Digital currency	Digital currency	Digital currency	General

Figura 2.5: Confronto dei linguaggi per gli smart contract di alcune delle principali blockchain [54]

Come si può vedere nella figura 2.5 , Ethereum ha molti linguaggi per lo sviluppo di smart contract, il principale è *Solidity*. Per i contratti in EOS.IO si utilizza C++. Si può anche notare che entrambi i linguaggi usati sono Turing-completi. Brevemente, un linguaggio è Turing-completo (TC) quando:

- Ha la capacità di implementare qualsiasi funzione calcolabile;
- Include sempre una funzione che non terminerà da sola;

- Include una funzione che, in teoria, potrebbe utilizzare una quantità infinita di memoria

Viene invece definito Non Turing-completo (NTC) quando non supporta loop, ricorsioni o altre varianti di go-to che tendono a non terminare da sole [35]. Fondamentalmente, un linguaggio Non Turing-completo è più specializzato, mentre un linguaggio Turing-completo è uno strumento multiuso in grado di gestire diverse situazioni. I linguaggi TC hanno applicazioni più ampie, mentre quelli NTC hanno usi specifici.

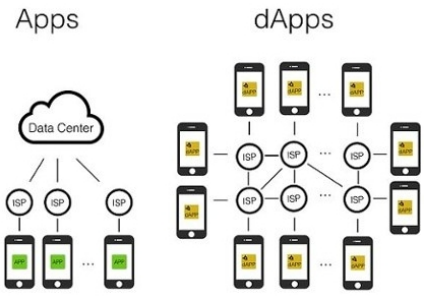
Affinché un'applicazione possa essere considerata decentralizzata deve rispettare i seguenti criteri:

- applicazioni completamente open-source;
- dati e record delle operazioni storicizzati su una blockchain pubblica;
- utilizzo di un token crittografico;
- generazione di token da parte dell'applicazione.

Una DApp viene classificata anche in base all'utilizzo o meno di una blockchain proprietaria [46]:

- tipo 1: DApp con una propria blockchain (Ethereum);
- tipo 2: DApp che utilizza la blockchain di una DApp di tipo 1 ma con token proprietario per il proprio funzionamento;
- tipo 3: DApp che utilizza il protocollo di una DApp di tipo 2.

La caratteristica principale è il codice decentralizzato, ovvero un codice ospitato su migliaia di computer ed eseguito successivamente in parallelo. Nel momento in cui un nodo della rete va down o si rifiuta di eseguire il codice, lo stesso codice viene eseguito su tutti gli altri computer (nodi) della rete.



Le DApp sono quindi considerate più flessibili, trasparenti, distribuite e resilienti. Una DApp è costituita da due parti principali: il lato back-end e lato front-end. Il lato back-end comprende l'insieme degli Smart Contract e rappresenta il cuore dell'applicazione dato che ne determina la logica e il relativo stato. Ogni modifica dello stato, attuata dall'esecuzione di una funzione contenuta nel codice degli Smart Contract, viene registrata nella blockchain.

2.4 EOS.IO: introduzione

La blockchain EOS.IO è una piattaforma open source di nuova generazione, leader per velocità di transazione e flessibile al punto tale da adattarsi alla maggior parte degli utilizzi.

EOS.IO è progettata non solo per casi d'uso di livello aziendale, ma è adatta a distribuzioni sia pubbliche che private. La piattaforma è personalizzabile in modo da soddisfare un'ampia gamma di esigenze aziendali, soprattutto nei settori con un sistema di autorizzazioni basato sui ruoli e con la necessità di un'elaborazione sicura delle transazioni offerte dalle applicazioni.

La creazione di applicazioni distribuite su EOS.IO segue schemi di sviluppo familiari e linguaggi di programmazione "non domain specific" utilizzati per lo sviluppo di applicazioni non blockchain. Per gli sviluppatori di applicazioni, la familiarità con l'ambiente di sviluppo si traduce in un'esperienza utente senza interruzioni, poiché consente agli sviluppatori di utilizzare i propri strumenti di sviluppo preferiti.

La piattaforma EOS.IO fornisce funzionalità come: account, autenticazione, database, comunicazione asincrona e pianificazione delle applicazioni su più core e cluster di CPU [11].

EOS.IO introduce una nuova architettura blockchain, progettata per consentire la scalabilità, sia verticale che orizzontale, delle applicazioni decentralizzate. La tecnologia risultante è un'architettura blockchain che alla fine può scalare fino a milioni di transazioni al secondo che consente una manutenzione delle applicazioni decentralizzate rapida e semplice [10]. Questo è un punto di forza, dato che in alcuni casi un'applicazione potrebbe non funzionare nel momento in cui viene raggiunta una soglia critica di utenti. Quin-

di, una piattaforma in grado di gestire un numero molto elevato di utenti è fondamentale.

Oltre al supporto delle operazioni di più utenti, EOS.IO si impegna nel migliorare la user experience offerta da altre blockchain. Nel momento in cui un utente effettua un'operazione, vorrebbe avere un feedback positivo della transazione avvenuta, nel minor tempo possibile. Ritardi più lunghi di qualche secondo frustrano gli utenti e rendono le applicazioni basate su una blockchain meno competitive rispetto alle alternative esistenti non blockchain. La piattaforma EOS.IO supporta una bassa latenza delle transazioni.

2.4.1 Delegated Proof of Stake

EOS fa uso di una variante della proof of stake, la delegated proof of stake (DPoS). Quelli che nella proof of work venivano indicati come miners, nella DPoS sono delegati o validators (validatori). I delegati in EOS sono 21 e vengono scelti tramite voto.

Il compito dei delegati non e più denominato mining (minare), ma viene sostituito dai termini minting/forging (coniare/forgiare). Per diventare un delegato occorre mettere una certa quantità di token bloccati in rete, questo procedimento è chiamato *stake*.

La quantità di token messi in stake aumenta le possibilità per un utente del sistema di essere scelto come delegato (figura 2.6). Ma non basta, per diventare un delegato occorre riceve il voto dagli altri utenti del sistema. Chiunque può scegliere di partecipare alla produzione di blocchi e avrà l'opportunità di produrli, a condizione che altri utenti utilizzino i propri token per avvalorare la loro candidatura [19].

1 token equivale a 1 voto ed essere in possesso di n tokens equivale ad esprimere n voti. Il compito dei delegati è quello di decentralizzare la rete. Un aspirante delegato decide anche quanto spazio di archiviazione dare alla rete e la quantità fornita, influenza il numero di voti che può ricevere.

I delegati fanno parte del gruppo di block producer, ed il numero di block producer è circa una sessantina, ma soltanto chi riceve più voti dalla community entra a far parte dei 21 delegati.

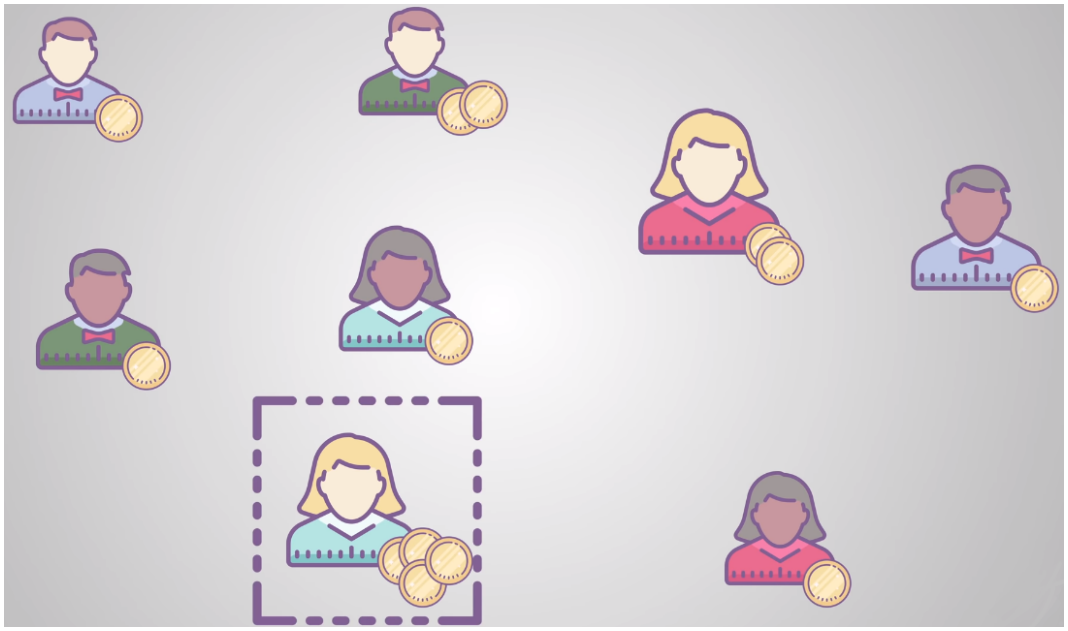


Figura 2.6: Delegati in EOS [28]

EOS.IO consente di produrre blocchi ogni 0,5 secondi e solo un delegato è autorizzato a produrre il blocco in un dato momento. Se non viene prodotto nel lasso di tempo programmato, il blocco per quell'intervallo di tempo viene saltato [10].

Al contrario della PoW, nella DPoS non vi è una competizione tra i miners nel validare un blocco, infatti, un delegato ha a disposizione una quantità di tempo per processarne uno e uno solo.

Nel momento in cui un delegato salta la validazione di un blocco, o non produce alcun blocco per 24 ore, oppure opera in maniera errata, si presentano due casistiche [10]:

- un nodo che aveva dato il suo voto al delegato può decidere di cambiare la sua preferenza e destinarlo ad un altro block producer;
- viene automaticamente rimosso, con la possibilità di rientrare a far parte dei delegati notificando l'intenzione di produrre nuovamente. Questo avviene nel caso in cui sia stato rimosso poiché non aveva prodotto blocchi.

Questo sistema permette di ridurre al minimo il numero di blocchi persi a causa dei produttori che si sono dimostrati inaffidabili, garantendo che la rete funzioni senza problemi.

Ogni transazione include un hash dell'header del blocco precedente per due motivi [10]:

- impedisce di inserire una transazione su fork che non includono il blocco a cui si fa riferimento;
- segnala alla rete i movimenti di un utente e su quale diramazione della blockchain sta operando.

In molte blockchain esistenti spesso si parla di *fork*. Fork è il termine che si utilizza per identificare diverse diramazioni della blockchain. Questo succede nel momento in cui più competitor riescono a trovare la soluzione del puzzle algoritmico in modo tale da aggiungere più di un blocco all'ultimo presente nella blockchain.

La DPoS subisce raramente fork, perché, piuttosto che competere, i produttori di blocchi cooperano tra di loro. In caso di fork, il consenso passerà automaticamente alla catena più lunga.

Questo metodo funziona perché la velocità con cui i blocchi vengono aggiunti a un fork di blockchain è direttamente proporzionale alla percentuale di produttori di blocchi che condividono lo stesso ramo della blockchain.

In termini più semplici, se ci sono più delegati che lavorano su una diramazione, essa crescerà più velocemente rispetto alla controparte. Un delegato non può produrre blocchi su due diramazioni della blockchain, pena l'esclusione. Motivo per cui ha più senso cooperare tra delegati. Dunque, tramite questo meccanismo di cooperazione, lo scopo di ridurre il numero di blocchi persi è facilmente raggiungibile.

Il sistema di storage utilizzato da EOS.IO è basato su un IPFS (InterPlanetary File System)[14], che è un protocollo p2p per la condivisione dei dati di un file system distribuito. I file sono riferiti tramite tabelle hash distribuite.

2.4.2 Smart contracts e accounts

Un account è identificato da un identificatore univoco ed il suo possesso è un requisito per interagire con la blockchain EOS.IO. A differenza della maggior parte delle altre criptovalute, i trasferimenti vengono inviati a un nome di account invece che a una chiave pubblica. Le chiavi attribuite all'account vengono utilizzate per firmare le transazioni [11].

EOS.IO permette di dare un nome all'account umanamente leggibile lungo 12 caratteri che viene scelto dal creatore dell'account. L'utente deve riservare la RAM richiesta per salvare il nuovo account.

In EOS una transazione viene chiamata anche azione, una transazione al suo interno può contenere ulteriori azioni. Ogni account può inviare un'azione ad un altro account, questa azione è definita in uno script. L'account deve anche avere uno script che gestisca le azioni quando esse vengono ricevute.

EOS.IO mette a disposizione di ogni account un proprio database privato che può essere acceduto dai propri action handlers. Gli script che gestiscono le azioni possono a loro volta inviare azioni ad altri accounts. L'unione delle azioni inviate e la gestione delle azioni ricevute è la definizione di smart contract per EOS.IO, i quali utilizzano C++ come linguaggio di programmazione.

Figura 2.7: Smart contract [31]

Gli account possono anche definire diversi *scope* all'interno del proprio database, dove per scope si intende una particolare organizzazione del database per rendere l'accesso ai dati più fluido. Ad esempio, ordinare i dati in base ad una chiave primaria oppure in base ad altri attributi della tabella.

Attraverso queste definizioni, i block producers schedulano le transazioni in modo tale che non vi sia conflitto di accesso. In questo modo si ottiene l'esecuzione parallela delle transazioni, che è uno dei miglioramenti tecnologici che offre EOS per aumentare il livello di performance (attualmente l'esecuzione parallela è solo un aspetto tecnico non ancora messo in pratica).

EOS.IO fornisce anche un sistema di gestione dei permessi per determinare se un'azione è autorizzata. L'autenticazione e la gestione dei permessi sono

separati tra loro. Questa scelta è stata presa per permettere agli sviluppatori di ottimizzare le performance per conto proprio.

La gestione dei permessi è organizzata in modo gerarchico. Il sistema permette agli account di definire quali combinazione di chiavi e account possono inviare una particolare azione ad un altro account.

In EOS.IO, gli account possono definire livelli di permessi ai quali è possibile attribuire un nome e ognuno di essi può essere derivato da livelli di permessi superiori ad esso. Ogni livello di permesso definisce un'autorità che serve per far sì che un'azione sia controllata allo stesso modo per tutti gli altri account che rispettano quel livello di permesso.

Inoltre, è consentito anche creare un mapping tra un contratto o un'azione di un altro account con il proprio permesso di livello denominato *permission mapping*. Ad esempio, se un utente decide di mappare una sua applicazione con il suo gruppo, ogni utente appartenente a quel gruppo può utilizzare l'applicazione. Per usufruire delle funzioni dell'applicazione, però, ogni utente utilizza la propria chiave. Ciò vuol dire che è sempre possibile identificare l'utente che ha usato l'applicazione e sapere cosa ha fatto.

Nella tecnologia implementata da EOS.IO tutti gli account hanno un gruppo denominato "owner", che può compiere qualsiasi azione, e un gruppo chiamato "active", per svolgere le principali transazioni, eccetto cambiare l'owner group. Tutti i gruppi di permessi derivano da "active".

Come si può vedere dalla figura 2.8, ad ogni permesso, nel caso in particolare in active, è possibile associare una tabella delle autorizzazioni. Ogni riga della tabella, indica un account o una particolare chiave pubblica per autorizzare azioni e transazioni di quell'account o chiave.

L'uso principale (e cruciale) del gruppo "owner", è la possibilità di recuperare la chiave di "active" nel caso in cui essa venga smarrita.

Un account di EOS.IO può fare il deploy di un solo contratto, pagando la quantità di token necessaria per mantenere il contratto in rete.

Nella figura 2.8, possiamo inoltre vedere, che nella tabella delle autorizzazioni vengono utilizzate due keyword: *threshold* e *weights*. La loro utilità è di poter creare degli account *multisignature*, il loro funzionamento è simile a quello dei normali account di EOS.IO.

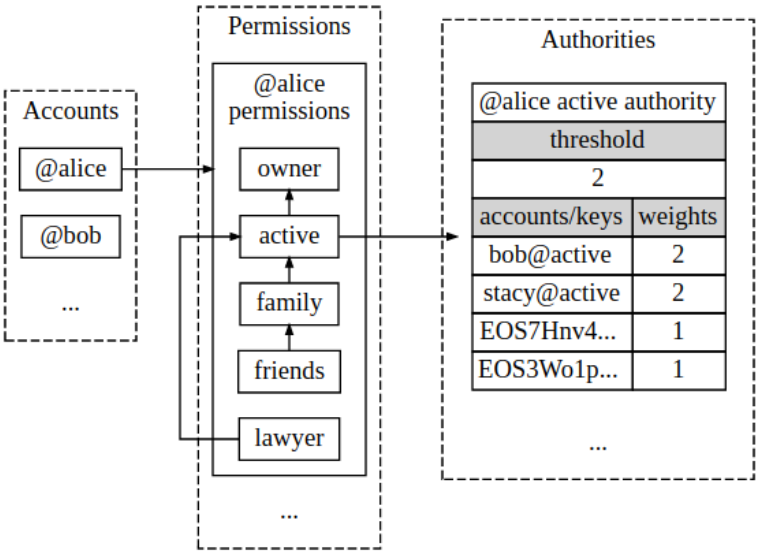


Figura 2.8: Struttura gerarchica dei permessi di un account in EOS.IO

In un normale account EOS, tutte le autorizzazioni hanno un threshold di 1 e hanno almeno una chiave con un weight di valore 1. In un account *multisignature* di EOS, il threshold è pari a 2 o superiore e vi sono almeno due chiavi ognuna con un weight variabile. Ciò significa che per effettuare una transazione con l'account *multisignature* sono necessarie più autorizzazioni [15].

2.4.3 Risorse di sistema: RAM, CPU e NET

In EOS il costo di una transazione verte sulle tre risorse principali di un account EOS: RAM, CPU e NET. Ad un utente viene richiesto di depositare dei token sul conto e di conseguenza distribuire una determinata quantità di token tra le tre risorse. La combinazione di CPU e NET costituiscono la bandwidth. Più alto è il numero di token che si mettono in "stake", più bandwidth si ha a disposizione sulla rete.

La RAM può solo essere acquistata e venduta; lo spazio posseduto serve per mantenere tutto ciò che viene salvato dall'account nella blockchain , a partire dallo smart contract fino alle righe aggiunte alle tabelle. EOS utilizza il Protocollo di Bancor [29] per gestire il prezzo della RAM sul mercato in base

alla domanda e all'offerta.

La CPU viene consumata ogni qualvolta viene eseguita una transazione sulla rete, ed il suo consumo è legato alla complessità del codice eseguito, quindi, più è lunga e complessa la transazione, più CPU verrà consumata.

La network corrisponde alla quantità dei dati inviati nella transazione e quindi viene consumata quando si esegue una transazione.

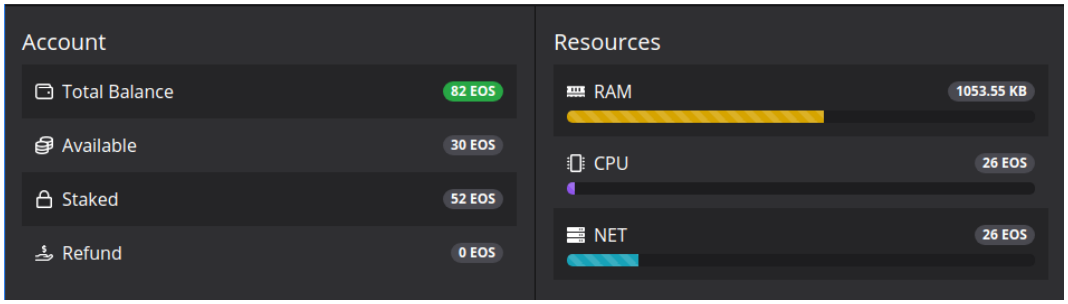


Figura 2.9: Statistiche di un account in EOS.IO

Quando si vogliono mettere in stake dei token per CPU e NET, occorre distribuire la giusta proporzione in base all'utilizzo che si farà nei successivi tre giorni. Infatti, ogni tre giorni la bandwidth viene "liberata" rendendo possibile il riutilizzo della quantità totale allocata [13].

Questo tipo di gestione, serve unicamente a controllare le azioni che un utente può fare, in modo da evitare che un nodo possa effettuare un numero incontrollato di transazioni.

Il meccanismo citato, non funziona con la risorsa RAM. L'unico modo per liberarla è cancellare le informazioni conservate all'interno dell'account: smart contract e righe di tabelle.

Per quanto riguarda invece il meccanismo di incentivo, EOS.IO ricompensa con nuovi EOS tutti i nodi che si prendono cura della rete periodicamente e ogni volta che validano un blocco.

Capitolo 3

Confronto tra EOS.IO e Ethereum

Le varie blockchain esistenti possono essere raggruppate in diverse generazioni, in base alle funzionalità offerte dalla piattaforma. Ad esempio, fanno parte della prima generazione piattaforme utilizzate principalmente per lo scambio di valori, come Bitcoin.

In questo capitolo, ci focalizziamo sulle caratteristiche offerte dalle piattaforme appartenenti alla seconda generazione (come Ethereum), che permettono la costruzione di applicazioni decentralizzate (DApps) attraverso la scrittura di smart contract. In particolare, analizzeremo le caratteristiche e le principali differenze che esistono tra due diverse blockchain di seconda generazione che hanno riscosso maggiore successo: Ethereum e EOS.IO.

3.1 Panoramica

Ethereum è stata presentata nel 2015 con lo scopo di creare un protocollo alternativo per la costruzione di applicazioni decentralizzate, creando così la seconda generazione di protocolli blockchain in contrasto con Bitcoin. EOS.IO, presentato nel 2018, invece si pone come alternativa, con l'obiettivo di andare incontro ai bisogni degli sviluppatori e agli utenti finali, ampliando con nuovi standard il DLT (Distributed Ledger Technology) per gli investitori.

EOS, come tutte le altre blockchain, si pone come principio di risolvere il *trilemma* [41] della scalabilità al centro del design delle blockchain (figura 3.1).

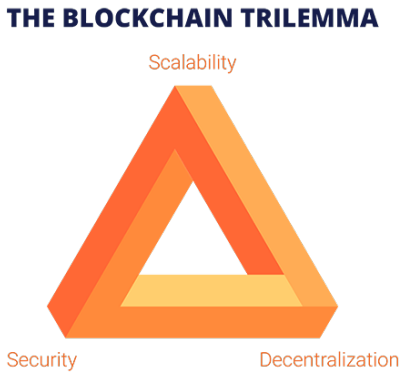


Figura 3.1: Trilemma di Vitalik Buterin [21]

Il trilemma ci pone davanti ad un bivio. Afferma Vitalik Buterin che è possibile ottenere solo due delle tre seguenti proprietà:

- scalabilità: un numero elevato di transazioni al secondo;
- decentramento: un gran numero di attori che partecipano alla produzione di blocchi e al processo di convalida;
- sicurezza: è costoso ottenere il controllo della maggioranza sulla rete.

Ethereum, la seconda rete più grande per capitalizzazione di mercato, favorisce il decentramento e la sicurezza a discapito della scalabilità.

Per quanto riguarda EOS.IO, presenta ottime caratteristiche di scalabilità e sicurezza ma poca decentralizzazione. Infatti, la piattaforma prevede di utilizzare per la produzione di blocchi un numero di 21 delegati. Tuttavia, non è detto che questo debba essere considerato come svantaggio, soprattutto se chi fa parte della rete è interessato a godere della scalabilità che EOS fornisce.

Ethereum tenta di risolvere il trilemma, perché la scalabilità è maggiore di quella di Bitcoin, ma non ancora sufficiente. EOS aumenta di molto la scalabilità, però la governance del sistema è nelle mani di un numero ristretto di delagati, per cui il livello di decentralizzazione è basso. Quindi nessuna delle due risolve a pieno il trilemma.

Nella seguente tabella (3.1) riassuntiva [32] sono indicate le principali differenze tra le due blockchain. Di seguito, analizzeremo quelle che sono individuate come pietre miliari delle blockchain, o per meglio dire quelle che concettualmente sono le differenze tecniche che le distinguono. Teoricamente, le differenze non sono altro che il risultato ottenuto dall'algoritmo di consenso utilizzato combinato con il protocollo di consenso.

	Ethereum (ETH)	EOS (EOS)
Fees	Configurable GasPrice Per Instruction	Per instruction
Incentive	2.0 ETH/MinedBlock + uncles	Configurable by BP
Block production rate	10-19 sec	0.5 sec
Confirmation time	1 min	1 sec
TPS	15	1000+
Languages	Solidity	C++
Interchain	No	Yes
Consensus algorithm	PoW (ethash)	Delegated PoS
Enhanced technology	Patricia Trees, Sharding	Merkle Proofs, Segwit
Fault Tolerance	51% ether attack	33% malus block producers
Permission schema	Permissionless	Permissioned

Tabella 3.1: Tabella riassuntiva delle principali differenze tecniche analizzate nelle blockchain EOS e Ethereum

3.2 Algoritmo di consenso

Proof of work, proof of stake e delegated proof of stake, sono solo alcuni dei meccanismi di consenso utilizzati dalle svariate blockchain.
Ethereum fa uso della PoW (proof of work), mentre EOS.IO della DPoS (delegated proof of stake).

Sebbene sia Ethereum che Bitcoin utilizzino l'algoritmo PoW, il funzionamento non è identico.
Riprendendo quello che è il processo chiamato mining, nel caso di Bitcoin i miners devono risolvere un puzzle crittografico molto complicato, usando la funzione hash (SHA-256) che prende in input tre valori [25]:

- una sequenza binaria detta *nonce* (che un miner deve trovare e che rispetti le proprietà imposte da Bitcoin);
- le transazioni del blocco che si vuole aggiungere alla catena;
- il valore della funzione hash del blocco precedente.

Per risolvere questo puzzle complicato è necessario disporre di molta potenza di calcolo, ciò vuol dire possedere una macchina molto costosa per trovare il *nonce* e di conseguenza aggiungere il blocco alla blockchain per ottenere una ricompensa in criptomonete.

Ovviamente, non tutti i nodi della rete hanno a disposizione una simile potenza di calcolo, inoltre un comune PC al giorno d'oggi avrebbe una percentuale di successo molto vicina allo zero.

Inoltre, i miners più esperti condividono la loro potenza di calcolo sulla rete con altri nodi in modo tale da risolvere il puzzle dividendo il lavoro; questa procedura è chiamata *mining pools*.

Lo scopo di Ethereum è dare le stesse probabilità a tutti i nodi. Tramite la funzione hash *Ethash*, Ethereum vuole combattere i chip ASIC (Application-Specific Integrated Circuit), ovvero quei componenti creati appositamente per risolvere i puzzle di produzione dei blocchi. Da quando Ethereum ha introdotto la sua funzione di hashing nella sua proof of work, anche i nodi meno potenti hanno potuto migliorare la loro potenza di calcolo tramite l'acquisto di schede

grafiche (GPU), le quali, pur non essendo specializzate, riescono comunque a competere con la potenza di calcolo di un'unità ASIC [33].

Come anticipato, EOS fa uso di una variante della proof of stake, la delegated proof of stake (DPoS) e che il concetto di miners viene sostituito da quello di delegati. I delegati in EOS sono 21 e vengono scelti tramite voto. Gli utenti che hanno un token possono votare. Inoltre, maggiore è il numero di token che si possiede, più spazio si potrà usare sulla rete.

A questo punto è lecito chiedersi perché scegliere un determinato algoritmo di consenso piuttosto che un altro. Parlando delle due tipologie, sono state evidenziate alcune differenze. Nella DPoS (2.4.1) non abbiamo più i miners, non vi è competizione e chiunque può ambire a diventare un delegato. Non abbiamo più un puzzle crittografico da risolvere, anzi, per velocizzare la creazione di nuovi blocchi, a turno i delegati hanno il compito di validarne uno in un certo periodo di tempo.

Figura 3.2: Differenze della potenza di calcolo dei nodi. PoW vs DPoS [28]

Facendo riferimento a queste distinzioni è semplice notare che anche in termini di energia i due algoritmi hanno consumi diversi. Nella proof of work (figura 3.2 a sinistra), un solo nodo della rete, nel minare un blocco, necessita di una potenza di calcolo rilevante. Considerando di avere la maggior parte dei nodi in competizione per minare un blocco, il dispendio energetico non passa inosservato. Per quanto riguarda la DPoS (figura 3.2 a destra), abbiamo solo i 21 delegati, che non richiedono una potenza di calcolo simile a quella dei

miners nella PoW. Sicuramente anche il consumo energetico è un paramentro del quale bisogna tenere conto nella scelta dell'algoritmo di consenso. Basti solo pensare che le reti basate su PoW utilizzano tanta energia, quanta quella utilizzata da un piccolo paese come l'Austria [52].

3.3 Cryptocurrency

Per quanto riguarda le criptovalute, la moneta di Ethereum è chiamata Ether e il suo simbolo è ETH. L'Ether è divisa in sottounità (figura 3.3) i cui nomi sono presi da personalità note nel mondo della crittografia e delle criptovalute. Nella tabella sottostante viene identificato il "wei" come valore unitario [50].

Per quanto riguarda EOS.IO, la sua moneta è denominata EOS. A differenza di Ethereum non vi sono delle sottounità, bensì utilizza la precisione per definire prezzi minori o intermedi. Come si può vedere in [5], il token proprietario EOS nella *mainnet* è creato con una precisione pari a 4. Questa non è una precisione prestabilita. Ogni token creato nella blockchain, è identificato attraverso un nome da tre lettere e dalla sua precisione.

unit	wei
wei	1
kwei / ada / femtotether	1.000
mwei / babbage / picoether	1.000.000
gwei / shannon / nanoether / nano	1.000.000.000
szabo / microether / micro	1.000.000.000.000
finney / milliether / milli	1.000.000.000.000.000
ether	1.000.000.000.000.000.000

Figura 3.3: Sottounità di Ethereum [50]

In circolazione ci sono 936 milioni di EOS, al momento della scrittura, e una fornitura totale di 1,02 miliardi di token. Block.one aveva un'offerta iniziale (ICO) [4] di monete per EOS nel giugno 2017 ed è durata un anno, periodo che sostanzialmente è più lungo di molte delle ICO viste in quel periodo. Nel corso del tempo è stato raccolto un totale di 4,02 miliardi. Inizialmente i token sono stati ripartiti dando il 10% ai fondatori, mentre il 90% è stato distribuito tra gli investitori [1].

Figura 3.4: Grafico che mostra le variazioni di prezzo di EOS

3.4 Modalità Operative

Prima di entrare nello specifico, distinguiamo i partecipanti all'interno di una blockchain che possono essere classificati in due categorie: *writers* e *readers* [53].

Il writer ("scrittore"), è colui che fondamentalmente aggiorna lo stato della blockchain. Nello specifico, è coinvolto attivamente nel protocollo di consenso, ovvero si occupa di accumulare transazioni, raccoglierle in un blocco e inserirlo in coda nella blockchain, favorendo così la crescita della stessa. Come già detto, questo lavoro è chiamato mining e il miner è colui che esegue queste operazioni.

Un reader invece è un entità che, a differenza di un writer, si limita solo ad avviare il processo di transazione oppure leggere e analizzare la blockchain.

Distinguiamo due modalità: *permissioned* e *permissionless* che corrispondono al *permission schema* di una blockchain [32]. Le blockchain permissioned hanno lo scopo di autorizzare un numero limitato di readers e writers a far parte della blockchain. Vi è un entità centrale che decide quali nodi possono o

meno partecipare alle operazioni di scrittura o lettura.

In caso di permissioned blockchain (figura 3.5), solo alcuni nodi hanno dei privilegi sulla validazione. Una blockchain permissioned ha una struttura centralizzata col potere di prendere decisioni e controllo sul processo di validazione dei blocchi. Inoltre, queste blockchain sono più veloci, più efficienti in energia e semplici da implementare.

Le permissioned blockchain cambiano la prospettiva iniziale sulle blockchain. Le blockchain erano originariamente pensate per essere sistemi aperti, gratuiti e pubblici, ma una blockchain permissioned è effettivamente l'opposto.

Queste richiedono l'autorizzazione per partecipare. Di conseguenza, il proprietario ha la capacità di stabilire chi può e chi non può entrare a far parte della sua rete. Questo controllo significa anche che il proprietario della blockchain può definire la struttura della rete, emettere aggiornamenti software e in generale controllare tutto ciò che avviene sulla sua blockchain [23].

Le transazioni su una permissioned blockchain sono convalidate solo dai membri approvati di quella blockchain. Il proprietario può anche controllare chi è autorizzato a vedere le informazioni relative. In alcuni casi gli utenti possono visualizzare una parte delle informazioni archiviate sulla catena di blocchi.

A causa della loro natura restrittiva e alla dimensione minore, le permissioned blockchain tendono ad essere più scalabili. Sono spesso più *centralizzate* rispetto alle permissionless blockchain. In effetti, il modello autorizzato è diventato popolare tra banche, supermercati, compagnie di navigazione e società di telecomunicazioni.

Alcune blockchain di questa tipologia, implementano la privacy permettendo ai nodi di essere eseguiti anche su altre blockchain parallele e interconnesse (interchaining).

Nel caso di pubbliche, permissionless blockchain (figura 3.6), nessuna autorità ha più potere degli altri (come Bitcoin e Ethereum), e tutti hanno il diritto di validare una transazione. Sono dunque pubbliche e *decentralizzate*. Non è richiesta alcuna autorizzazione, ogni nodo può entrare a far parte della blockchain sia come reader che come writer. Non vi è un'entità centrale che gestisce i membri della blockchain o che può bandire un nodo. Questo non

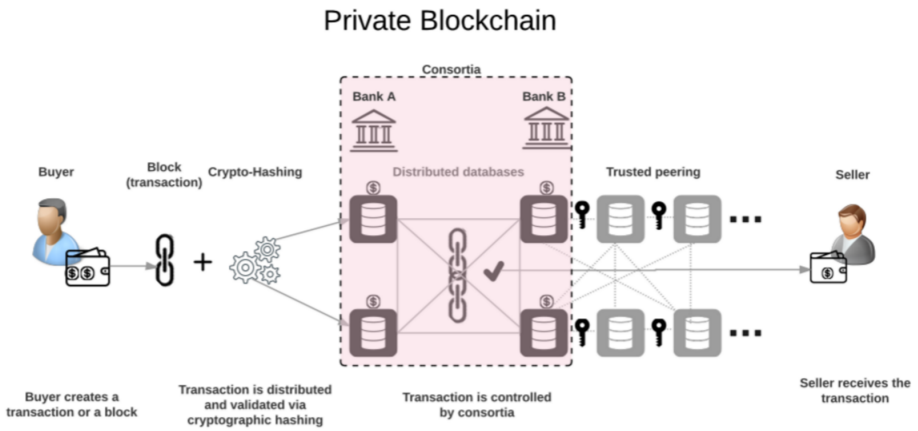


Figura 3.5: Funzionamento di una permissioned blockchain [43]

vieta comunque la possibilità, per un protocollo di consenso, di implementare un livello di privacy per nascondere informazioni sensibili dei nodi.

Molti considerano il modello permissionless blockchain più vicino al concetto originale di blockchain, come delineato da Satoshi Nakamoto.

Poiché chiunque può aderire a una permissionless blockchain, le stesse tendono ad essere molto più decentralizzate di un sistema permissioned. Un compromesso è che le blockchain senza autorizzazione sono spesso più lente delle alternative permissioned.

Le informazioni sulle transazioni archiviate sulle permissionless blockchain vengono solitamente convalidate dal pubblico. Senza terze parti per regolare ciò che accade, il sistema si basa su questo per raggiungere un consenso su quali transazioni sono considerate valide e sicure.

Identifichiamo Ethereum di tipo permissionless. Per quanto riguarda EOS, invece, viene definita come una *public permissioned* blockchain: pubblica perché chiunque può vedere le transazioni effettuate ed entrare a far parte della blockchian per poter effettuare transazioni; permissioned poiché il potere di validare le transazioni è affidato ai 21 delegati risultando scalabile.

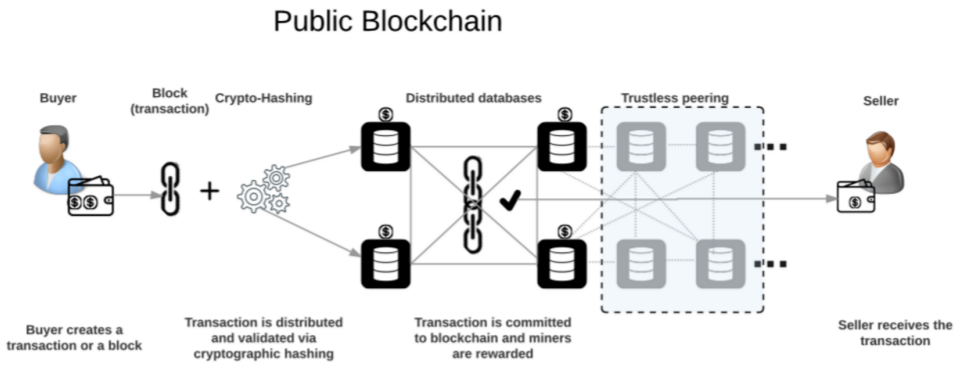


Figura 3.6: Funzionamento di una permissionless blockchain [43]

3.5 Centralizzazione e Decentralizzazione

Nel paragrafo precedente abbiamo accennato ai concetti di centralizzazione e decentralizzazione.

In teoria le blockchain non sono centralizzate poiché i dati vengono salvati e aggiornati in maniera decentralizzata. Tuttavia alcune blockchain implementano un certo grado di centralizzazione. Ad esempio in Bitcoin il mining, per quanto potenzialmente paritario, ormai è gestito da pochissime mining pools.

In generale, decentralizzazione significa che nessuna singola entità ha il controllo su tutta l'elaborazione.

Confrontare centralizzazione e decentralizzazione è come paragonare un database ad una blockchain: un database è conservato in un determinato luogo e la modifica dei record può essere apportata solo da chi ha il potere di farlo senza la necessità di consultare qualcun altro. Al contrario, in una blockchain, per effettuare una modifica bisogna avere il consenso dalla maggior parte dei partecipanti (figura 3.7).

Uno dei principali vantaggi di una blockchain decentralizzata è la possibilità di contrastare i nodi malevoli. Ad esempio, se un nodo o un gruppo di nodi prova a manipolare la blockchain, altri partecipanti della blockchain possono esprimere il loro disaccordo e bloccare la modifica dei dati.

L'effetto dei meccanismi di incentivo nella decentralizzazione è molto delicato, ed è strettamente legato con l'algoritmo di consenso adottato nella block-

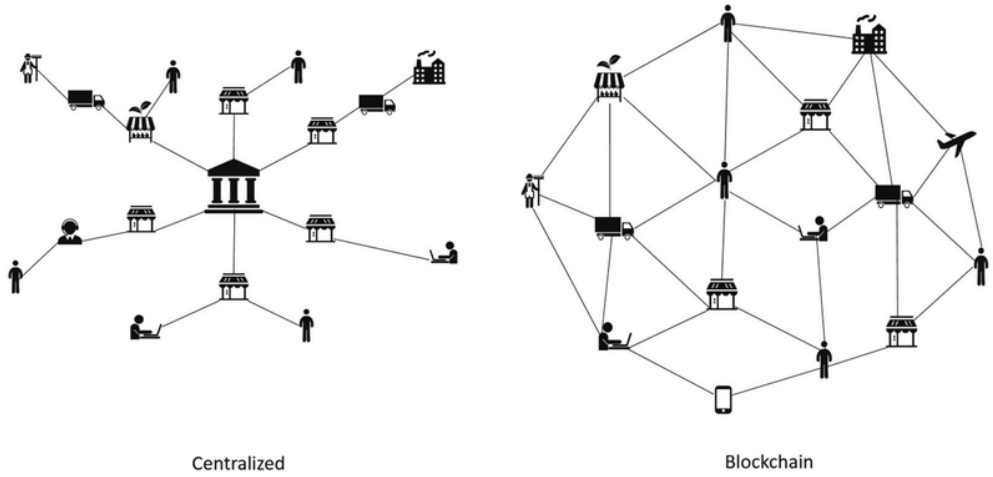


Figura 3.7: centralizzazione vs decentralizzazione delle blockchain [49]

chain. DPoS è centralizzato per scelta progettuale, PoW tende a essere centralizzato di fatto. Uno dei principali problemi dei protocolli che implementano algoritmi come la proof of work, è il mining pool.

Ethereum ad esempio, è per metà gestita da 4-5 mining pool che si occupano di minare i blocchi. Così, unendo le potenze di calcolo e gestendo tutte le transazioni che vengono aggiunte ai blocchi, la blockchain risulta essere per l'80% gestita da queste pool [38]. D'altro canto la DPoS è centralizzata per scelta di progetto, per questo i block producer (delegati) di EOS.IO sono solo 21.

Più l'incentivo attira, più i nodi vengono creati, ciò implica più opportunità per la decentralizzazione di coesistere con il protocollo di consenso, anche se centralizzato. Quindi la decentralizzazione è in stretta relazione con:

- incentivi;
- protocollo di consenso;
- permission schema;
- implementazione del libro mastro (ledger).

3.6 Costi

Teoricamente cominciare ad utilizzare una blockachain è gratuito, ma ci sono tre aspetti che impongono un costo sulla rete:

- costo per le transazioni da effettuare (fees);
- minimo costo fisso per il deploy degli smart contract;

Questi costi possono diventare pesanti quando vi è una congestione della rete, ma eliminarli del tutto potrebbe permettere attacchi malevoli al sistema in modo gratuito.

Anche in questo caso Ethereum e EOS si differenziano. L'idea di Ethereum è di far pagare una tassa ogni qualvolta viene effettuata una transazione oppure quando viene fatto il deploy di uno smart contract. Ethereum fa utilizzo del concetto di gas. Per gas si intende un meccanismo per limitare la complessità dell'esecuzione di uno smart contract.

La somma di tutte le operazioni effettuate determina un costo complessivo, in più l'utente che fa il deploy dello smart contract, assegna un costo unitario detto gas price che va a determinare la tassa da pagare; più alta è la tassa, più velocemente verrà validata la transazione dai miners che riceveranno come ricompensa il costo del gas speso. Invece un utente che si presta a effettuare una transazione può scegliere il limite di unità di gas che vuole spendere.

Se il limite è troppo basso si rischia che la transazione venga rifiutata e il gas speso viene perso. Se invece il limite è alto e si manda più gas di quanto ne serva, allora il resto viene rimborsato [50].

In sostanza, è meglio fornire una maggiore quantità di gas per eseguire una transazione. Nel momento in cui si va a eseguire una transazione quindi viene presentato un massimo costo che potrebbe essere pagato, non quello che si pagherà effettivamente.

Abbiamo già parlato di come vengono affrontati i costi in EOS.IO (sezione 2.4.3). Il deploy dei contratti e l'esecuzione delle transazioni si basa sulle risorse: RAM, CPU e NET. Come detto, CPU e NET vengono rimborsate ogni tre giorni dall'esecuzione delle transazioni. Questa scelta è paragonabile al gas

di Ethereum. Lo scopo è di non permettere ai nodi di eseguire gratuitamente delle operazioni, prevenendo così una buona parte degli attacchi possibili alle blockchain come quelli di tipo DDoS.

3.7 Consistenza

Per consistenza si intende confermare che una transazione è stata aggiunta in modo sicuro alla blockchain. Un blocco, contenente transazioni, nel momento in cui viene confermato, viene detto *irreversibile* o immutabile.

Il metodo utilizzato dalle attuali blockchain è aspettare che venga aggiunto un certo numero di blocchi successivi dopo quella transazione. Viene considerato come un trade-off tra tempo e sicurezza. Questi due fattori sono in stretto contatto tra loro, ogni blockchain ha le sue tempistiche in base alle regole che propone per aggiungere un blocco alla catena.

La consistenza è dunque una funzione del tempo di conferma che dipende anche dal BPR (Block Production Rate), che rappresenta il tempo necessario per minare un blocco. In realtà, nessuna attuale implementazione può assicurare una consistenza del 100%. Più il tempo passa, più vengono confermati blocchi, dunque più è alta la probabilità che quella transazione sia immutabile.

In Ethereum, una transazione, per essere considerata consistente, necessita di 12 blocchi successivi nella catena.

In EOS, un blocco si può considerare irreversibile nel momento in cui si raggiunge una maggioranza tra i delegati che confermano un blocco. La maggioranza richiesta è pari ai $2/3 + 1$ dei delegati [12].

Nella DPoS, tipicamente, abbiamo una completa partecipazione dei block producer. In tal modo una transazione può considerarsi confermata nel 99,9% dei casi con una media di $0,25$ secondi. Inoltre, EOS.IO aggiunge la Byzantine Fault Tolerance asincrona (aBFT) [26] per ottenere rapidamente l'irreversibilità. L'algoritmo aBFT fornisce una conferma del 100% dell'irreversibilità entro 1 secondo [10]. In seguito, vedremo che Jungle, la testnet che abbiamo utilizzato e una delle più performanti, impiega circa 90 secondi per far si che un blocco diventi irreversibile.

Questo risultato sembra una contraddizione con quanto introdotto nella tabella 3.1, in realtà, questa differenza dimostra solo una discrepanza tra la teoria e la pratica.

3.8 Performance

Per quanto riguarda la performance occorre valutare latenza e scalabilità. Il bottleneck delle blockchain è la latenza, cioè il tempo necessario per ottenere il consenso di una transazione. Infatti il principale problema di Ethereum è il tempo di attesa, ovvero il tempo impiegato per la validazione di un blocco, che comprende l'elaborazione di un nuovo blocco, mining e la conferma dei blocchi precedenti.

Per scalabilità ci si riferisce al numero di TPS (Transactions Per Second) che una blockchain è capace di raggiungere attraverso le sue regole combinate con l'algoritmo di consenso utilizzato. Quando si parla di scalabilità si considerano: la dimensione dei dati sulla blockchain, la frequenza con cui vengono elaborate le transazioni e la latenza di trasmissione dei dati.

L'algoritmo di consenso combinato con il protocollo utilizzato gestiscono, in base alle esigenze o meno, scalabilità e performance. La Proof of Work non offre una buona scalabilità e anche basse performance, mentre altri protocolli, come la DPoS, offrono buone performance con una ottima scalabilità (figura 3.8). Ad oggi non è chiaro quale sia il miglior trade-off tra numero di nodi e prestazioni.
Quindi performance e scalabilità sono influenzati da:

- TPS (Transactions Per Second);
- BPR (Block Prodution Rate);
- protocollo di consenso;
- permission schema;

Negli anni, in Ethereum, si sono verificate congestioni di rete nelle quali una transazione, per essere validata, deve attendere in coda.

	Ethereum	EOS.IO
TPS (#transaction)	15-25	4000
TPS giornaliere	600 K (record 1mln)	50 mln (record 70mln)
BPR (seconds)	10-19	0,5-1
Block dimension	20-30 KB	1 MB

Figura 3.8: Confronto performance Ethereum vs EOS.IO

L'alternativa per un utente è aumentare la probabilità che la sua transazione venga minata più velocemente aumentando il costo del gas associato ad essa. Storicamente, si è verificato che il costo per ottenere conferma in tempo ragionevole è arrivato a circa duemila dollari [40].

Per quanto riguarda EOS invece, è improbabile che si possa verificare una situazione simile data la sua alta scalabilità.

Per quanto riguarda la dimensione dei blocchi, Ethereum non ha una dimensione fissa dei blocchi.

La dimensione del blocco di Ethereum è vincolata dal numero di unità di gas che possono essere contenute per blocco. Questo limite è detto *limite del gas di blocco*, il quale non ha niente in comune o con il limite del gas di transazione.

I miners accettano blocchi con un limite medio di gas per blocco di circa 10 milioni di gas. Dunque, la dimensione di un blocco Ethereum è compresa mediamente tra i 20 e 30 KB di dimensione.

Durante la convalida di una transazione, i miner di Ethereum determinano quale dovrebbe essere il limite massimo di gas di blocco, segnalandolo in rete. I miners possiedono questa capacità di regolare il limite del gas perché le modifiche influiscono sulle risorse necessarie per minare efficacemente sulla blockchain Ethereum.

Quindi, non è possibile dire a priori il numero di transazioni che possono entrare a far parte di un blocco, poiché dipende dalla complessità, e in modo correlato, al costo della transazione. Le transazioni che sono più complesse richiedono più spazio di archiviazione sulla blockchain constando più gas. In

generale, più operazioni esegue una transazione, maggiore sarà il suo costo in gas. In teoria, qualcuno potrebbe spendere tutto il limite di gas di un blocco in una transazione [16].

In EOS, invece, abbiamo una dimensione di 1 MB. Anche qui i block producer possono cambiare la dimensione del blocco, ma ciò richiede una *hard fork* nel protocollo.

Un hard fork è un cambiamento radicale al protocollo di una rete che rende validi blocchi e transazioni precedentemente non validi, o viceversa. Un hard fork richiede che tutti i nodi o gli utenti eseguano l'aggiornamento alla versione più recente del software del protocollo [18].

3.9 Funzionalità e Estensibilità

L'obiettivo delle blockchain è supportare il pagamento elettronico e decentralizzato basato sulla crittografia piuttosto che sulla fiducia. L'idea è di applicare la tecnologia anche alle DApps per la registrazione del nome e uso di token in ambito aziendale.
Quindi quando si parla di funzionalità e estensibilità, ci si riferisce al supporto di implementazioni complesse della blockchain con funzionalità built-in o attraverso la possibilità di implementare estensioni che permettono agli sviluppatori di assemblare applicazioni al di là dell'intento originale della piattaforma usata. Nelle funzionalità built-in vi sono ovviamente la gestione delle transazioni e del sistema di crittografia.

Una delle funzionalità che si vuole implementare è la comunicazione tra diverse catene: consentire a diverse blockchain di interagire mantenendo invariate le loro proprietà di sicurezza. L'obiettivo che si vuole raggiungere è permettere a blockchain con diversi scopi di interagire e trarne benefici l'una dall'altra. Questa proprietà è chiamata *interchaining*.

Molte blockchain forniscono un supporto per le interazioni con altre blockchain, una di queste è EOS che usa il Merkle Proof [24] per la validazione dei client. Al contrario, Ethereum, non nasce con lo scopo di favorire l'interchain.

A volte potrebbe essere necessario usare degli adapters (adattatori) per collegare due implementazioni diverse di blockchain e permettergli di comunicare.

3.10 Accounts

In Ethereum, ogni account è univocamente rappresentato da un indirizzo di 20 byte. Un account può essere controllato da un utente o spedito come smart contract [9]. Ci sono due tipi di account:

- externally owned: controllato da un utente con delle chiavi private;
- contract: è uno smart contract spedito alla rete e l'esecuzione è controllata tramite il codice.

Entrambi i tipi di account hanno l'abilità di ricevere, trattenere e inviare ETH o token, e interagire con altri smart contract.
Le differenze sono le seguenti: un externally owned account può creare un account senza pagare alcun costo; può avviare una transazione; le transazioni tra externally owned accounts possono trasferire solo ETH. Per i contratti invece, creare l'account ha un costo poiché si usa dello spazio in rete, l'account può inviare transazioni solo in risposta ad una transazione ricevuta e, inoltre, una transazione ricevuta da un external account può innescare il codice per eseguire diverse azioni, come ad esempio trasferire token o creare nuovi contratti.
Un account è formato da 4 campi:

- il nonce: un contatore che indica il numero di transazioni inviate dall'account. Assicura che una transazione sia stata processata una sola volta. Se è un account di tipo contratto, questo numero rappresenta il numero di contratti creati dall'account;
- balance: saldo corrente del conto;
- code hash: si riferisce ai frammenti di codice contenuti nel database sotto il loro corrispondente valore in hash. Nel caso in cui è un contract account, allora, conterrà il suo valore in hash con cui verrà salvato. Per gli externally owned accounts vi è l'hash della stringa vuota;
- storage root: è un valore hash a 256 bit del nodo radice del Merkel Patricia tree[riferimento]. È vuoto di default.

Per quanto riguarda gli account in EOS facciamo riferimento alla sezione 2.4.2.

3.11 Sicurezza

I problemi di sicurezza delle blockchain riguardano bugs o vulnerabilità che possono essere sfruttate per lanciare un attacco. Se un dato viene approvato da tutti i nodi non può essere rimosso dal ledger (neanche modificato), fornendo così sicurezza e integrità dei dati.

Attualmente la Proof of Work è la più sicura, nonostante abbia il 51% di possibilità di essere attaccata [32].

Uno dei principali attacchi delle blockchain permissionless è il *selfish mining*: un miner mina un blocco, ma non lo manda in broadcast e prova a minarne un altro facendo una catena personale e pubblicare la catena successivamente tutta di un colpo. In questo modo, ottiene ricompense maggiori rispetto a quanto dovrebbe effettivamente riceverne.

Altri protocolli come proof of stake o DPoS forniscono, come già spiegato, migliori performance e/o scalabilità, ma questo implica un trade-off a livello di sicurezza: tollerano fino a 1/3 dei nodi maliziosi. Questo risultato viene raggiunto aggiungendo anche alcuni livelli di centralizzazione della rete.

Altri algoritmi sopportano fino a 2/3 di nodi maliziosi, ma richiedono delle restrizioni. Ad esempio, una delle tecniche conosciute è accettare solo *trusted nodes* cioè nodi fidati. Questa tecnica consiste nel fare in modo che, per entrare a far parte della rete, un nodo interno deve fornire garanzie per quello entrante. Serve per testimoniare che questi nodi si conoscono a vicenda, garantendo sulla sicurezza degli altri.

La sicurezza di una blockchain è influenzata da delle decisioni tecniche che riguardano:

- fault tollerance;
- algoritmo di consenso;
- permission schema.

3.12 Privacy

La maggior parte delle blockchain sono progettate per proteggere l'integrità della transazione, senza però considerare la privacy della transazione stessa. Le blockchain godono della privacy quando: le transazioni non possono essere collegate tra loro; solo i partecipanti conoscono il contenuto della transazione. Nel nostro caso, sia Ethereum che EOS.IO implementano un certo grado di *pseudo-anonimità* per quanto riguarda l'esecuzione delle transazioni, poiché lo stesso account viene utilizzato ripetutamente per effettuare transazioni. Sebbene in EOS.IO si possa scegliere un nome leggibile piuttosto che randomico, come per Ethereum, questo viene considerato come un grado di pseudo-anonimità dato che la descrizione del nome dell'account non è riconducibile ad un utente in particolare.

Nelle blockchain private, la trasparenza della cronologia delle transazioni non è un requisito [32]. È desiderabile la trasparenza per le applicazioni, o per semplicità, aggiungere un livello di controllo per proteggere i dati della blockchain. La privacy è condizionata da:

- protocollo di consenso (definisce cosa i nodi/miners possono vedere);
- i miglioramenti tecnologici che una blockchain può decidere di implementare.

Capitolo 4

Decentralized Rating Framework

Questo capitolo ha lo scopo di descrivere la struttura complessiva e le funzionalità richieste dal Decentralized Rating Framework, focalizzando l'attenzione sull'organizzazione logica del sistema, gli attori presenti, e l'interazione tra le varie entità.

Inoltre, permette una comprensione della struttura e del funzionamento del sistema attraverso una descrizione delle dipendenze dei vari moduli e delle eventuali operazioni che è possibile eseguire.

La sezione 4.1 ha l'obiettivo di illustrare al lettore la nomenclatura utilizzata nell'ambito dei sistemi di recensione, permettendo la comprensione del linguaggio tecnico utilizzato nei paragrafi successivi. La sezione 4.2 mostra i vantaggi determinati dall'introduzione dei token nel sistema; la sezione 4.3 presenta l'architettura del sistema esponendo le funzionalità dei vari moduli; la sezione 4.4 mostra il funzionamento del sistema attraverso la descrizione del flusso delle operazioni eseguite in uno scenario di riferimento.

4.1 Sistema di recensione: introduzione

Le entità coinvolte in un sistema di recensione possono essere raggruppate in tre grandi categorie:

1. **Utenti**: sono i clienti del sistema. Ogni utente può interagire con il sistema effettuando recensioni per un certo item dopo aver fatto un acquisto. Ad ogni recensione effettuata, l'utente aumenta la propria skill relativa alla tipologia di item recensito. Un comportamento attivo da parte dell'utente contribuisce al miglioramento del sistema fornendo un numero di input maggiore alle funzioni che si occupano del calcolo del rating.

2. **Item**: sono i servizi o le attività che possono essere recensite. Il sistema di recensione si preoccupa di memorizzare tutte le recensioni relative ad un item, per poi effettuare il calcolo del rating quando un utente lo richiede. Gli item rappresentano il punto di contatto tra il proprietario di un'attività e il sistema di recensione. Permettono di reperire i feedback degli utenti contribuendo al miglioramento dell'attività o del servizio.

3. **Operazioni**: rappresentano l'interfaccia che il sistema di recensione espone agli utenti. In altre parole, rappresentano l'interazione che ogni utente può avere con il sistema e quindi l'insieme delle operazioni che può svolgere all'interno di esso. Ogni utente ha la possibilità di recensire, ricercare un certo item secondo un criterio e visualizzare il rating assegnato ad un item. L'operazione che caratterizza un sistema di recensione è ovviamente il rating.

4.2 Token del sistema

All'interno della blockchain è definito un token principale di riferimento utilizzato per i pagamenti. Data la possibilità di creare un token alternativo, nel nostro ecosistema il token principale è stato denominato "RSF" (Rating System Framework).

All'interno del nostro sistema di rating, la creazione di nuovi token è vista come l'utilizzo di coupon messi a disposizione dall'item che li crea. Lo scopo è permettere all'utente che li possiede di effettuare i pagamenti con l'equivalente del token principale dell'ecosistema in termini di coupon.

I coupon vengono guadagnati dall'utente come compenso per aver recensito l'item. Per determinare il numero di coupon da trasferire è stato introdotto il concetto di skill. Una skill è una competenza che l'utente acquisisce interagendo attivamente con gli items che richiedono quella particolare skill. Ad ogni item è associata una skill che lo caratterizza; un utente può accrescere le sue competenze (molteplici skills) recensendo gli items.

Dunque il valore dei coupon trasferito per ricompensare l'utente, è pari al punteggio che possiede l'utente nella skill che caratterizza l'item.

Nel nostro sistema di recensione, il token viene quindi utilizzato come risorsa economica per correlare l'esperienza dell'utente alle sue skill.

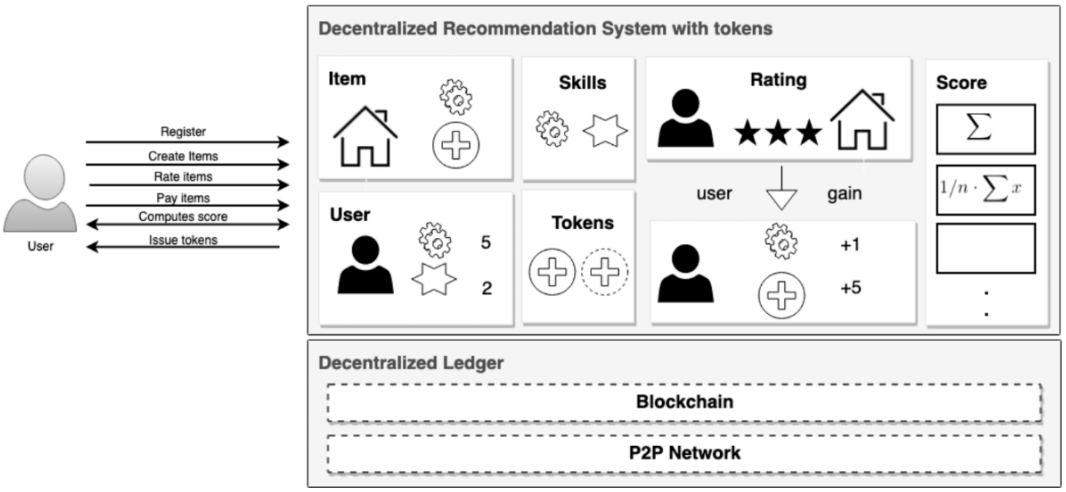


Figura 4.1: Overview del sistema di recensione con tokens e skills

Rappresenta, inoltre, un incentivo sia per gli utenti che per i proprietari degli item infatti:

- Gli utenti sono incentivati a recensire gli item dato che la pubblicazione della recensione comporta un aumento della loro abilità nel sistema e una ricompensa in token;
- I proprietari degli item sono incentivati a premiare gli utenti che hanno una maggiore esperienza poiché le recensioni lasciate da essi risultano essere più influenti e affidabili.

Il framework che abbiamo sviluppato fornisce diversi benefici ed è caratterizzato dalle seguenti proprietà:

1. Pubblico: gli items e le recensioni degli utenti sono pubblici (visibili a tutti gli utenti e replicati su tutti i nodi della blockchain);

2. Decentralizzato: il framework si basa su un ambiente decentralizzato dove non esiste un'autorità centrale che gestisca le recensioni, le tariffe e il punteggio di elementi;

3. Tamper-proof: I dati del sistema (recensioni, funzioni di calcolo, skills, tokens) memorizzate nella blockchain non possono essere alterati;

4. Disponibilità e Persistenza: i dati del sistema memorizzati sulla blockchain sono sempre disponibili, e non possono essere rimossi.

5. Token economy: possibilità da parte degli utenti di essere retribuiti per le loro recensioni.

4.3 Architettura del Sistema

Il nostro sistema è diviso in due principali moduli: uno per la gestione degli utenti, items e per tutte le varie operazioni messe a disposizione di chi usufruisce dell'applicazione; l'altro per la gestione dei token.

4.3.1 Modulo RatingSystem

Il RatingSystem è il contratto principale dell'applicazione, mette a disposizione le operazioni (azioni) che interfacciano gli utenti con la DApp. Le azioni che un utente può effettuare sono:

- creare un nuovo utente all'interno del sistema che abbia il nome di un account presente sulla blockchain;

- disattivare un account;

- aggiungere un nuovo item alla piattaforma specificando qual è la skill che lo caratterizza, il nome e la quantità di coupon che vuole creare, e il valore che il coupon possiede rispetto alla moneta principale del Rating System;
- effettuare il pagamento di una fattura specificando il codice, la quantità da pagare e la volontà di pagare o meno con il token/coupon dell'item;
- recensire un item inserendo come parametri il codice del pagamento effettuato in precedenza e la valutazione dell'item;

Un utente che è invece possessore di un item, ha a disposizione le seguenti operazioni:

- disattivare l'item che ha creato;
- inserire una fattura da pagare, specificando il codice, il cliente e la quantità di token da pagare.

Inoltre, il gestore del sistema ha a disposizione l'operazione che consente di aggiungere delle nuove skills al sistema, utili per caratterizzare nuovi items. In più, è possibile invocare, tramite un'azione, una delle funzioni che raccolgono un insieme di recensioni e restituisce un determinato punteggio a seguito di un noto criterio di calcolo.

4.3.2 Modulo gestore dei token

Il contratto che gestisce i token si occupa del trasferimento di denaro da un account ad un altro facendo uso delle opportune azioni.

L'obiettivo è modellare il contratto per utilizzarlo in modo appropriato all'interno della piattaforma per lo scambio di token/coupon che vivono nel sistema di recensione.

In sostanza nel sistema il token assume il significato di risorsa economica e di conseguenza può essere utilizzato come forma di pagamento. Ogni token viene identificato da un insieme di caratteristiche:

- il simbolo che lo rappresenta;

- l'offerta iniziale.

Quando un utente decide di pubblicare un item deve necessariamente specificare le varie caratteristiche del token da utilizzare. La scelta del valore del coupon rispetto al token del Rating System (RSF), risulta fondamentale da un punto di vista economico dato che determina la competitività dell'item nel sistema.

Il contratto mette a disposizione delle operazioni, alcune di queste entrano a far parte direttamente della logica del contratto RatingSystem. Un esempio è l'azione di trasferimento di denaro, la quale risiede nel contratto di gestione dei token e viene invocata dal modulo RatingSystem.
Le azioni implementate dal modulo di gestione dei token sono:

- creazione di un token;
- emissione del token all'account che ne ha richiesto la creazione;
- possibilità di ritirare i token messi in circolazione;
- trasferimento dei token da un utente A ad un utente B;
- possibilità di creare un account che possiede un bilancio pari a zero di un determinato token;
- possibilità di eliminare un token.

4.4 Funzionalità del sistema

Di seguito andremo ad elencare le principali funzionalità del sistema attraverso dei diagrammi di sequenza, prendendo in esame gli scenari di:

- creazione di un utente e di un item;
- fase di pagamento.

4.4.1 Creazione users e items

Nello scenario presentato vedremo come gli utenti dovranno interagire con il sistema per la creazione di un user e di items all'interno della piattaforma (figura 4.2). Per descrivere la sequenza utilizziamo due classici attori: Alice e Bob.

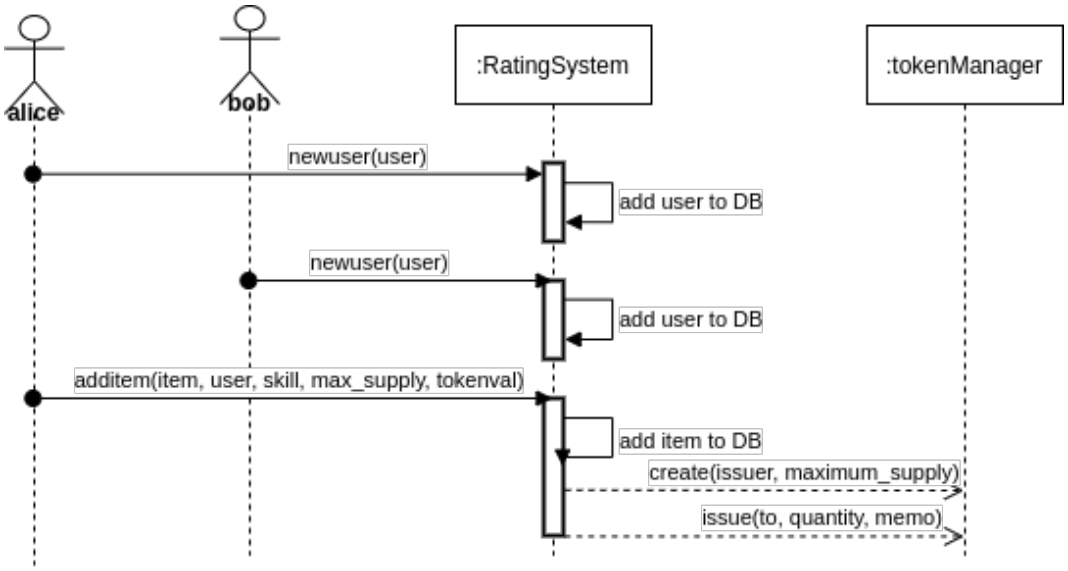


Figura 4.2: Diagramma di sequenza che mostra come avviene la creazione di due account all'interno del Rating System Framework

1. Alice vuole entrare a far parte del sistema di rating. Attraverso l'azione *newuser* crea l'account alice. È richiesto che il nome dell'utente da creare, sia lo stesso dell'account che richiede la creazione. L'account può pagare e recensire.

2. Alice, proprietaria di un ristorante, vuole aggiungere il suo locale sulla piattaforma per permettere ai suoi clienti di pagare, recensire ed essere ricompensati con dei coupon che lei stessa chiederà di emettere. Attraverso l'azione *additem*, Alice dovrà fornire il nome del suo ristorante, il suo nome come proprietaria, la quantità di coupon che vuole creare e il valore del coupon rispetto al token della piattaforma (RSF).

4.4.2 Fase di pagamento

In questo scenario presenteremo la fase di pagamento utilizzando sempre Alice e Bob come attori che interagiscono con il sistema. Alice è dunque proprietaria di un item, mentre Bob è un semplice cliente .

La situazione in cui ci poniamo è quella relativa alla figura 4.2, dove Alice e Bob possiedono un account, ma solo Alice ha inserito il suo locale tra gli items del sistema.

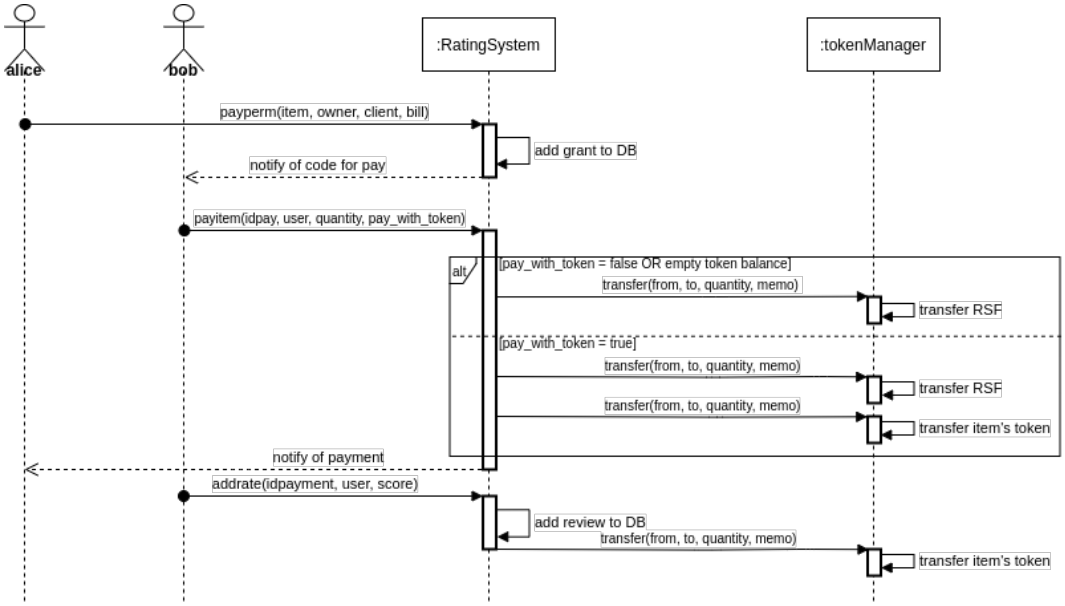


Figura 4.3: Diagramma di sequenza che mostra la fase completa di pagamento

1. Bob usufruisce del servizio offerto dal locale di Alice. Di conseguenza, Alice, autorizza l'account di Bob per il pagamento del servizio attraverso l'azione *payperm*.

2. Al seguito del permesso garantito, viene inviata una transazione di notifica a Bob, la quale informa del codice univoco utile per effettuare il pagamento e l'ammontare del costo del servizio.

3. In possesso dei dati ricevuti, Bob può effettuare il pagamento. È possibile, con l'azione *payitem*, decidere se pagare o meno con i coupon

ricevuti precedentemente. Se il pagamento avviene utilizzando solo il token RSF, ammesso che ce ne siano abbastanza, viene effettuato un solo trasferimento. Se Bob decide di effettuare il pagamento tramite coupon, ovviamente, in base al numero di token/coupon avremo due casi:

- se il saldo del coupon è tale da permettere il pagamento totale della fattura, allora verrà effettuata una sola transazione;
- se il saldo non è abbastanza vengono effettuate due transazioni: una parte della fattura viene pagata in coupon, l'altra in RSF.

4. Al termine della transazione, Alice riceverà una notifica di avvenuto pagamento.

5. Solo a pagamento avvenuto, Bob potrà recensire l'item tramite l'azione *addrate*, al termine del quale l'account di Alice trasferirà automaticamente i coupon sull'account di Bob.

4.4.3 Calcolo delle recensioni

Il sistema mette a disposizione delle funzioni che permettono di avere una valutazione del servizio offerto da un item in base alle recensioni ottenute in passato.

Le funzioni implementate sono la media semplice e la media pesata. Un utente che vuole conoscere la media dei voti fatte da altri clienti durante la recensione, deve utilizzare una delle azioni che preferisce tra: *avg* e *weightedavg*. Il risultato verrà restituito tramite notifica.

Capitolo 5

Implementazione Rating System in EOS.IO

In questo capitolo mostreremo l'implementazione del sistema di recensione definito nel capitolo 4 prendendo come architettura di riferimento la piattaforma EOS.IO. A seguito della struttura della sua piattaforma, in EOS.IO, l'implementazione dei contratti ha subito una variazione sostanziale: in Solidty (Ethereum) è facile gestire nuove istanze di un oggetto quali nuovi utenti e nuovi items; in C++, usando le apposite librerie di sviluppo di smart contract per EOS.IO, la gestione delle informazioni da memorizzare varia drasticamente, tanto da arrivare a creare un vero e proprio database.

5.1 Librerie di EOS.IO

5.1.1 eosio.hpp

La principale libreria di sviluppo a disposizione è *eosio.hpp*. Questa libreria consente, attraverso delle macro, di vedere una classe come un contratto, di specificare le azioni e le strutture dati. Ad esempio, come si può vedere nel codice 5.1, l'uso delle parentesi quadre, specificando all'interno *eosio::contract*, serve come direttiva al compilatore per interpretare la classe che viene creata

come codice smart contract; dopo i due punti viene ereditata la classe contract di eosio.

```
class [[eosio::contract("RatingSystem")]] RatingSystem :
public eosio::contract
```

Codice 5.1: Uso della macro eosio::contract

Una sintassi simile la ritroviamo nella dichiarazione di un'azione (codice 5.2), dove nel dichiarare la funzione, prima di scrivere la classica *signature*, si utilizza la macro per specificare che quella determinata funzione è in realtà la dichiarazione di un'azione/operazione dello smart contract ove dichiarata.

```
[[eosio::action]] void newuser(const name& user);
```

Codice 5.2: Uso della macro eosio::action

La particolarità di uno smart contract in EOS.IO, è la possibilità di creare tabelle (5.3), le quali verranno riempite con le informazioni relative alle transazioni che saranno effettuate dagli utenti del sistema.

Dopo aver specificato i campi della tabella, è necessario definire una funzione utile per valutare la chiave all'interno della stessa, nel momento in cui viene effettuata una ricerca.

L'ulteriore innovazione è quella di poter definire tabelle multi-indice [11], cioè tabelle in grado di descrivere diverse funzioni di ricerca che non richiedono necessariamente che il campo per cui la ricerca viene effettuata sia una chiave primaria.

Nel codice 5.3 vi è la creazione di una struct, la quale tramite la macro *eosio::table*, viene vista come una tabella del database che utilizzeremo come struttura dati per tutte le informazioni che verranno salvate, relative alle diverse operazioni fatte interagendo con lo smart contract. Affinché la tabella sia in grado di recuperare la chiave primaria, è necessario che l'oggetto memorizzato all'interno della tabella abbia una funzione membro *const* chiamata *primary_key()* che restituisce un intero senza segno a 64 bit.

```
struct [[eosio::table]] items
{
  name iname; //*PK

  name owner; //*Foreign Key
  name skill; //*Foreign Key
  symbol sym;
  double tokenval;
  bool active;

  uint64_t primary_key() const { return iname.value; }
  uint64_t by_secondary() const { return owner.value; }
  uint64_t by_tertiary() const { return skill.value; }
};
```

Codice 5.3: Uso della macro eosio::table

Invece, in 5.4, viene instanziata la tabella appena creata come una tabella multi_index. In tal modo è possibile legare le funzioni definite per le ricerche, diverse da quella di chiave primaria, come metodi di ricerca all'interno della tabella.

```
typedef eosio::multi_index<
  "items"_n, items,
  indexed_by<"byowner"_n,
    const_mem_fun<items, uint64_t, &items::by_secondary>>,
  indexed_by<"byskill"_n,
    const_mem_fun<items, uint64_t, &items::by_tertiary>>
>itemsTable;
```

Codice 5.4: Uso della struttura multi_index

Inoltre, la libreria permette di definire quali azioni sono utilizzabili sotto forma di *inline action*. Una inline action è un'azione del contratto stesso o messa a disposizione da un altro smart contract che è possibile invocare e utilizzare. Attraverso il seguente codice (figura 5.5) è possibile dichiarare che l'azione è invocabile da un contratto esterno. In questo caso, la direttiva *using* serve per indicare con che nome la funzione deve essere chiamata esternamente; va

specificato il nome della funzione tra virgolette (in questo esempio "transfer") e va specificando dove risiede il codice in questione (nel contratto rsftoken).

```
using transfer_action =
    eosio::action_wrapper<"transfer"_n, &rsftoken::transfer>;
```

Codice 5.5: Uso di action_wrapper per poter usufruire di un'azione come inline action

5.1.2 asset.hpp

La libreria *asset.hpp* fornisce al programmatore la possibilità di interagire con i token creati all'interno del sistema. Abbiamo già parlato di come sono definiti i token in EOS.IO nel paragrafo 3.3, in cui abbiamo messo in evidenza come, oltre al nome, sia possibile dare una precisione ad un token. Il tipo *asset* consente di riferirci ad un token raggruppando le sue caratteristiche con una struttura contenente due campi (figura 5.1): un intero a 64 bit e un tipo *symbol*.

Type	Name
int64_t	amount
symbol	symbol

Figura 5.1: Tipo asset

Il primo campo, senza il secondo, non assume un particolare valore. Ciò che rende completa la rappresentazione del token è il tipo *symbol*, anch'esso della libreria *asset.hpp*. Questo tipo è composto da due campi: il codice o stringa di rappresentazione del simbolo e un intero senza segno a 8 bit che rappresenta la precisione del token.

$$\text{eg: } asset(1500000, symbol("EOS", 4)) => 150.0000\ EOS$$

5.2 Organizzazione del database

Come già preannunciato, attraverso l'utilizzo delle tabelle multi-index, in EOS.IO è possibile realizzare un vero e proprio database per la gestione delle informazioni relative al sistema di rating.

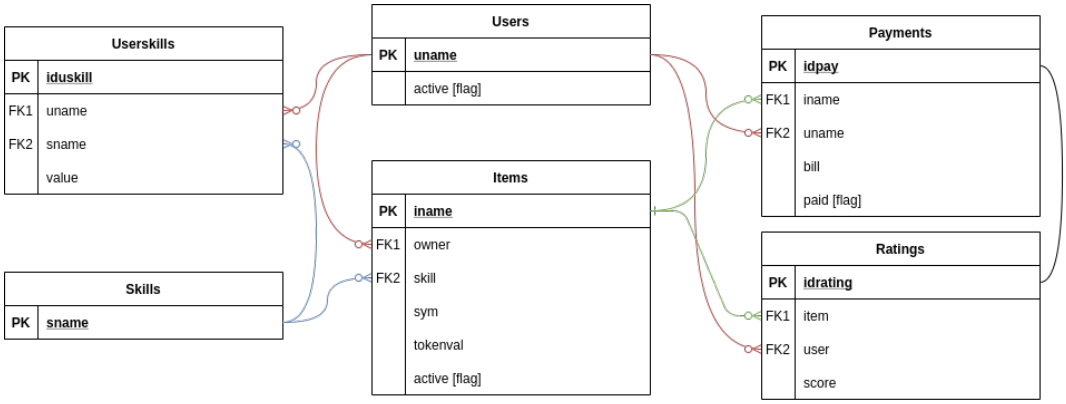


Figura 5.2: Raffigurazione del database per mantenere le informazioni su EOS.IO

Nella figura 5.2, abbiamo tutte le tabelle con le quali è possibile interagire tramite le azioni che il Rating System Framework fornisce.

La tabella *users* rappresenta la tabella degli utenti, contenente due campi: il nome dell'utente e un flag per comunicare se l'utente è attivo o meno. Così facendo si mantiene la consistenza dei dati nel caso in cui un utente non voglia più far parte del Rating System.

La stessa logica è stata applicata agli items del sistema. Nella tabella *items*, inoltre, abbiamo i campi relativi al nome dell'item, il proprietario, la skill che lo caratterizza, il simbolo del coupon/token emesso e il valore del token.

La tabella *skills* contiene tutte le skills, tra le quali un item owner può scegliere quale attribuire al proprio servizio. A sua volta la skill sarà legata anche all'utente nella tabella *userskills* con il valore di competenza che lo stesso raggiunge man mano che effettua delle recensioni.

Per registrare i pagamenti da effettuare e/o effettuati, abbiamo definito la tabella *payments*. I campi della tabella sono: un codice univoco per distinguere i pagamenti, il nome dell'item al quale effettuare il pagamento, il

nome del cliente che dovrà pagare, l'importo e un valore booleano che dice se il pagamento è stato completato o meno.

Il codice del pagamento viene importato nella tabella *ratings*, dato che per ogni pagamento eseguito è possibile inserire una sola recensione. Gli altri campi della tabella sono: l'item al quale viene fatta la recensione, il cliente che la effettua e il punteggio che viene assegnato.

5.3 Token management system

Il sistema realizzato possiede una gestione interna dei token utilizzati per effettuare i pagamenti. Il token di riferimento, quello utilizzato per effettuare i pagamenti di un cliente verso un determinato item, è il token denominato RSF (Rating System Framework). Dunque il token visto sotto un punto di vista del tipo *symbol*, è stato definito come $("RSF", 4)$, dove si ricorda che 4 è la precisione che si vuole assegnare al token.

Ogni item, può inoltre creare il suo token visto come coupon. Il problema principale della gestione dei token è determinato dalla discrepanza di valore di ogni coupon creato rispetto al token RSF. Se sia Alice e Bob possiedono un item, non è detto che entrambi vogliano dare lo stesso valore al proprio token. L'obiettivo è non avere perdite durante i pagamenti a cause di arrotondamenti per token con precisione diversa.

Ogni pagamento è contrassegnato con un valore in RSF. Se un cliente preferisce pagare in coupon, la conversione del coupon, se di precisione minore rispetto a RSF, rischia di richiedere arrotondamenti, i quali implicherebbero troncamenti di informazioni che non vorremo perdere. Per uniformare il sistema, ogni token deve essere creato con una precisione pari a 4.

5.4 Azioni

In questa sezione faremo una panoramica generale di tutte le azioni nello Smart Contract implementate nel Rating System Framework. Di ogni azione verrà data una breve descrizione, individuando chi è l'attore che può eseguirla e se la sua esecuzione comporta una modifica dello stato della blockchain, ovvero se la risorsa RAM viene consumata dall'account. Inoltre, illustreremo anche come si esegue una transazione utilizzando *cleos*, un tool a riga di comando per eseguire varie operazioni sulla blockchain tra cui anche il così detto *push* delle transazioni. Ometteremo l'utilizzo del flag *-u*, utilizzato per eseguire una determinata transazione collegandosi tramite un API endpoint, come ad esempio quello di una testnet:

-u https://jungle3.cryptolions.io:443

Il flag che invece non possiamo omettere per ogni azione è *-p*, che sta per *permission*; ovvero permette di specificare l'attore dell'azione e il permesso con il quale vuole effettuarla.

L'azione *addskill* (codice 5.6) è l'unica che può essere eseguita da un unico account, ovvero quello amministratore del sistema.

```
cleos push action ratingsystem addskill '["tradizionale"]'
    -p ratingsystem@active
```

Codice 5.6: Esecuzione dell'azione addskill usando cleos

Permette al gestore di inserire le skill che possono essere usate per descrivere le competenze dei clienti o per identificare la tipologia di item. Il parametro che deve essere passato è di tipo *name* di EOS.IO, quindi una stringa di massimo 12 caratteri legali. Aggiungere una skill all'interno della tabella del database comporta il costo della RAM.

In seguito abbiamo le azioni *newuser* (codice 5.7) e *deluser*. La prima serve per aggiungere un nuovo utente al database dando come paramentro il nome

dell'utente con la condizione che sia il nome di un account che appartiene alla blockchain.

```
cleos push action ratingsystem newuser '["alice"]'
    -p alice@active
```

Codice 5.7: Esecuzione dell'azione newuser usando cleos

La seconda serve per disattivare l'utente all'interno del sistema ponendo un valore booleano nello stato *false*. Questa scelta implementativa è stata presa per evitare di creare inconsistenze nel database. Entrambe le transazioni possono essere effettutate solo dall'utente in questione ed entrambe modificano lo stato della blockchain: newuser aggiunge una nuova riga, deluser cambia il valore di quella riga.

Altre due azioni sono *additem* (codice 5.8) e *delitem*. Un utente che fa già parte del Rating System e che possiede un item può effettuare queste azioni. Additem permette di aggiungere un item fornendo il nome dello stesso, il nome dell'owner, la skill che lo caratterizza, la quantità di coupon che si vogliono mettere a disposizione e il valore del coupon rispetto al token RSF.

```
cleos push action ratingsystem additem '["ristorante", "alice",
    "tradizionale", "100.0000 RIS", 0.3 ]' -p alice@active
```

Codice 5.8: Esecuzione dell'azione additem usando cleos

Delitem permette, invece, di disattivare l'item usando la stessa logica della deluser, passando come parametri il nome dell'item e il suo owner. Le due azioni modificano lo stato del database aggiungendo una riga per quanto riguarda additem, e modificando il valore booleano per quanto riguarda delitem.

Un item owner può utilizzare l'azione *payperm* (codice 5.9) per permettere ad un cliente di effettuare un pagamento. I parametri passati sono: il nome dell'item, l'owner, il nome del client e il prezzo da pagare come asset.

```
cleos push action ratingsystem payperm '["ristorante", "alice",
    "bob", "1.5000 RSF"]' -p alice@active
```

Codice 5.9: Esecuzione dell'azione payperm usando cleos

Questa transazione inserisce una riga all'interno del database, modificandone lo stato. Inoltre l'owner pagherà anche per l'invio della notifica di permesso concesso, contenente un codice per il pagamento, verso il cliente al quale è destinato il pagamento.

Successivamente, un cliente userà l'azione *payitem* (codice 5.10) per pagare un item. Inserirà come parametri della transazione il codice di pagamento ricevuto, il suo nome, la quantità da pagare ed avrà anche la possibilità di decidere se pagare con il token RSF oppure con i coupon accumulati precedentemente.

```
cleos push action ratingsystem payitem '[0, "bob",
    "1.5000 RSF", false]' -p bob@active
```

Codice 5.10: Esecuzione dell'azione payitem usando cleos

A pagamento effettuato viene modificato un valore booleano nella riga che attestava la richiesta del pagamento. Quindi il costo della riga non è più attribuito al item owner, bensì al cliente.

Solo dopo aver effettuato un pagamento, un cliente può aggiungere una recensione, tramite la *addrate* (codice 5.11), specificando il codice che è stato usato per il pagamento, il proprio nome e lo score che si vuole assegnare. Questa transazione comporta l'aggiunta di una riga nella tabella delle recensioni, pagando così della RAM per la modifica dello stato della blockchain.

```
cleos push action ratingsystem addrate '[0, "bob", 8]'
    -p bob@active
```

Codice 5.11: Esecuzione dell'azione addrate usando cleos

In seguito alla recensione effettuata, l'item owner deve ricompensare il cliente con i coupon. Anche lui, dunque, tramite questa transazione, andrà a modificare lo stato del database pagando della RAM.

Le ultime due azioni non vanno a modificare lo stato, sono le funzioni di valutazione delle recensioni: *avg* una media semplice e textitweightedavg una media pesata. I parametri da inserire sono: il nome dell'item del quale si vuole sapere la valutazione, ed il nome dell'utente che richiede l'esecuzione della transazione. Il nome dell'utente è necessario poichè dato che nessuna azione in uno smart contract può ritornare un valore, il valore calcolato dalla funzione verrà inviato come notifica all'account che lo richiede.

5.5 Notifiche

Al termine delle azioni *payperm* e *payitem* vengono mandate delle notifiche. Per la prima viene mandato il codice del pagamento e l'importo da trasferire, nella seconda viene notificato l'avvenuto pagamento di una fattura.

La notifica non è molto diversa dalle altre operazioni, è anch'essa un'azione e di conseguenza la sua chiamata all'interno del contratto è una inline action.

Prendiamo in esame l'invio di una notifica al termine di una *payperm*. Viene chiamata la funzione *send_notify* (5.12), passando come parametri il nome dell'utente ed il messaggio da notificare.

```
send_notify(client, "code: " + to_string(code)
    + ", bill: " + bill.to_string());
```

Codice 5.12: Chiamata della funzione send_notify al termine della'azione payperm

La funzione send_notify specificata nel codice 5.13, invia un'azione dando come parametri: il livello di permesso con il quale chiamare l'azione, dove risiede il codice e il nome dell'azione e, infine, i parametri passati come tupla.

```
void send_notify(const name& user, const string& message)
{
    action(
        //permission_level,
        permission_level{get_self(), "active"_n},
        //code,
        get_self(),
        //action,
        "notify"_n,
        //data
        std::make_tuple(user, message)
    ).send();
}
```

Codice 5.13: Funzione send_notify che performa la chiamata dell'azione notify

All'interno dell'azione (codice 5.14) l'invio della notifica viene eseguito tramite la funzione *require_recipient* che, come parametro, prende il nome dell'utente al quale inviare la notifica.

```
[[eosio::action]] void notify(
    const name& user,
    const string& message)
{
  require_auth(get_self());
  require_recipient(user);
}
```

Codice 5.14: Action notify che performa l'invio della notifica

Nella figura 5.3 sono raffigurati i permessi dell'account *alice* nell'IDE EOS Studio [7]. Come detto nella sezione delle autorizzazioni (2.4.2), è possibile inserire sia chiavi che account come autorizzazione. In figura, il permesso *ratingsystem@eosio.code*, serve per far si che l'account che ha fatto il deploy del sistema di rating, possa mandare le notifiche all'account.

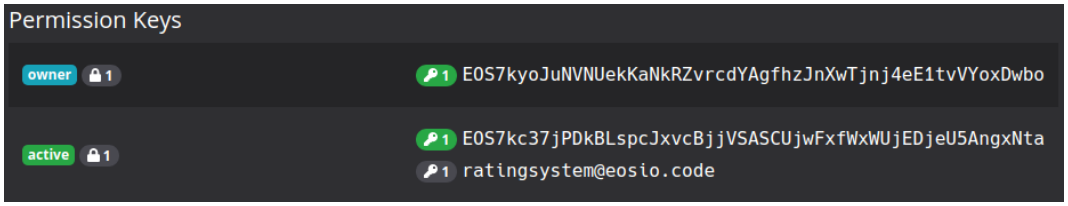


Figura 5.3: Rappresentazione grafica dei permessi in EOS Studio

```
auth={{
  "account": "alicersfacco",
  "permission": "active",
  "parent": "owner",
  "auth": {
    "threshold": 1,
    "keys": [{
        "key":
        "EOS8ZMSJDFCUSnkpAD2XP7rPr2QeCex8msMeLbUNkY9kn8y98GgTf",
        "weight": 1
      }],
    "accounts": [{
        "permission": {
          "actor": "ratingsystem",
          "permission": "eosio.code"
        },
        "weight": 1}],
    "waits": []
  }
}}
```

Codice 5.15: Json che Alice usa per aggiornare le sue autorizzazioni e permettere allo smart contract "ratingsystem" di inviarle le notifiche

Aggiungere un permesso, è un'operazione che può essere fatta in diversi modi: utilizzare una delle funzioni messe a disposizione dal tool di EOS per personalizzare l'account oppure utilizzare direttamente il contratto eosio, in particolare l'azione *updateauth*. Il payload di questa azione sarà un json avente una struttura come nel codice 5.15.

5.6 Compilazione e Deploy

Prima di effettuare il deploy del contratto all'interno della blockchain o all'interno di una testnet, è necessario compilare il contratto scritto in C++. Nella figura 5.4 è rappresentato il flusso di compilazione di uno smart contract in EOS.IO che termina con il deploy sulla blockchain del contratto stesso.

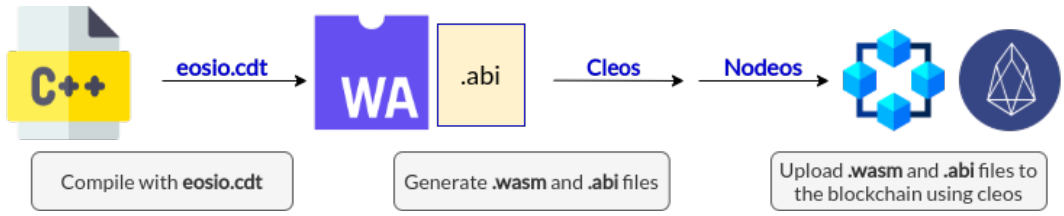


Figura 5.4: Fase di compilazione e deploy di un contratto

5.6.1 Compilazione

La fase di compilazione prevede l'utilizzo dell'apposito compilatore *eosio-cpp*. EOS.IO fornisce ai programmatori il Contract Development Toolkit, quindi è necessario installare *eosio.cdt*, all'interno del quale risiede *eosio-cpp*.

Questa collezione di tools è capace di interpretare le macro utilizzate per definire azioni o tabelle all'interno del contratto, generando, attraverso la compilazione il file ABI (Application Binary Interface).

Questo file è una rappresentazione in JSON delle azioni e del database realizzati nel contratto. Chiunque può leggere il file ABI è in grado di interagire con il contratto tramite JSON.

I parametri del comando (codice 5.16) sono molto simili ad una compilazione di un tipico programma scritto in C++.

```
eosio-cpp -abigen -I include -I ../rsf.token/include/
    -R resource -contract RatingSystem
    -o RatingSystem.wasm src/RatingSystem.cpp
```

Codice 5.16: Comando di compilazione del contratto RatingSystem

Come si può notare uno dei flag da inserire è *-abigen*, responsabile della creazione del file .abi. Di seguito vi è il flag *-I* per includere file di header (.hpp). Nel nostro caso, il contratto RatingSystem ha un suo file di header (RatingSystem.hpp situato nella cartella *include*) e necessita di includere anche il file di header di rsf.token per riconoscere le inline action. Di seguito si specifica il nome del contratto tramite il flag *-contract*, e con il flag *-o* si genera il file .wasm. Il file .wasm (WebAssembly bytecode) è un formato che permette di definire e importare funzioni che non sono dichiarate nel file stesso, bensì nell'ambiente in cui il codice viene eseguito.

5.6.2 Deploy dei contratti

Effettuare il deploy di uno smart contract consiste nell'utilizzo di un tool che consente di interfacciarsi con varie componenti (figura 5.5) della blockchain: *cleos*.

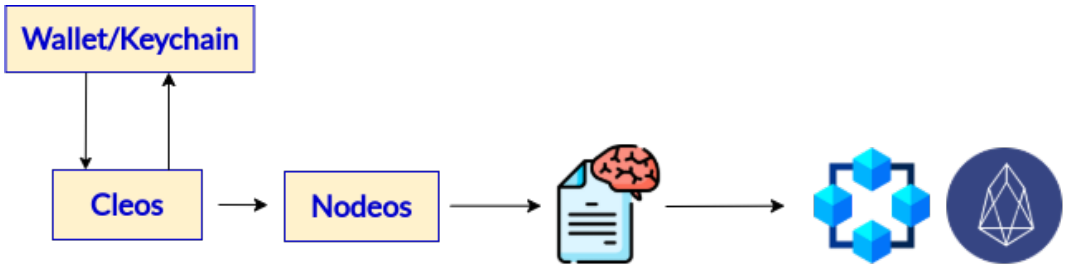


Figura 5.5: Componenti della piattaforma EOS.IO

Cleos è un tool a riga di comando che si interfaccia con l'API REST di *nodeos*. *Nodeos* è il componente di EOS.IO che viene eseguito su ogni nodo della rete. Può essere configurato per elaborare smart contract, convalidare transazioni, produrre blocchi contenenti transazioni valide e confermare blocchi per registrarli sulla blockchain.

Nel codice 5.17, è raffigurato un'esempio di avvio di nodeos nel quale vengono specificati vari flag per la configurazione del nodo. L'ultimo comando serve per testare se nodeos è stato avviato correttamente, dunque, se produce dei blocchi.

Gli sviluppatori possono anche utilizzare cleos per testare gli smart contract: consente di creare wallets, account, di importare le chiavi necessarie che serviranno per eseguire le transazioni e per fare il deploy degli smart contract.

```
nodeos -e -p eosio \
    --plugin eosio::producer_plugin \
    --plugin eosio::producer_api_plugin \
    --plugin eosio::chain_api_plugin \
    --plugin eosio::http_plugin \
    --plugin eosio::history_plugin \
    --plugin eosio::history_api_plugin \
    --filter-on="*" \
    --access-control-allow-origin='*' \
    --contracts-console \
    --http-validate-host=false \
    --verbose-http-errors >> nodeos.log 2>&1 \
    --fix-reversible-blocks \
    --hard-replay-blockchain &

curl http://localhost:8888/v1/chain/get_info
```

Codice 5.17: Esempio di avvio di nodeos con le varie configurazioni

Precedentemente, abbiamo detto che in EOS.IO un account può eseguire il deploy di un solo contratto (2.4.2). Avendo due moduli, abbiamo bisogno di due account.

In figura 5.6, vengono rappresentati i passaggi iniziali che un amministratore deve eseguire per inizializzare il sistema.

1. L'account *rsf* esegue il deploy dello smart contract contenente le operazioni disponibili e il database della DApp.

2. L'account *rsf.token* esegue il deploy dello smart contract *rsf.token* adibito alla gestione dei token all'interno della DApp.

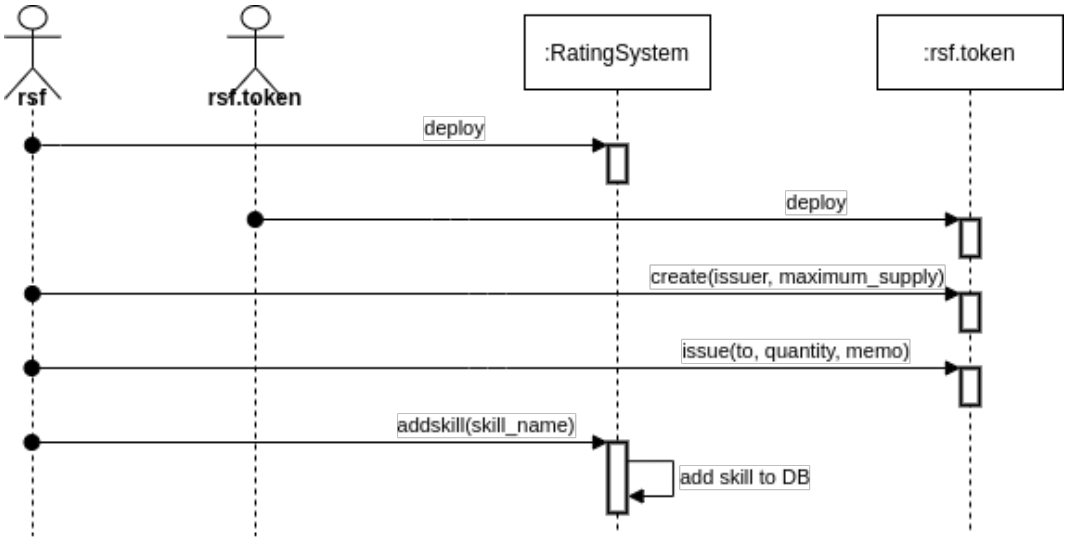


Figura 5.6: Diagramma di sequenza che mostra la fase di deploy dei contratti e le prime interazioni con essi

3. L'account *rsf* utilizza l'azione *create* del contratto *rsf.token* per creare il token principale dell'applicazione: *RSF*.

4. L'azione *issue* è necessaria per emettere il token, in tal modo *rsf* può gestire il suo token trasferendolo agli utenti che fanno parte del sistema.

5. L'ultimo passo per configurare il sistema è aggiungere delle skills tramite l'azione *addskill*, così gli utenti che aggiungeranno un nuovo item potranno utilizzare una delle skills a disposizione.

Eseguire il deploy di uno smart contract tramite i tool citati, vuol dire caricare sulla blockchain i file ottenuti in fase di compilazione: il file *.abi* e il file *.wasm*. Questa operazione avviene attraverso il nodo *eosio* della blockchain contenente le azioni *setcode* e *setabi*. L'account che usa implicitamente queste azioni accetta di riservare la RAM per lo smart contract del quale fare il deploy.

5.7 EOS Studio

Uno dei principali strumenti di sviluppo utilizzati è *EOS Studio*. EOS Studio è un IDE con funzionalità integrate come: compilazione di smart contract e deploy su una blockchain in locale, oppure su una testnet. Consente di creare istanze di nodeos e ingloba le funzionalità di cleos facilitando l'utilizzo tramite interfaccia grafica, soprattutto per quanto riguarda il push delle azioni, il controllo delle risorse degli account e i relativi permessi. Nel momento in cui si vuole fare il push di un'action, bisogna inviare le informazioni tramite json. Utilizzare EOS Studio permette di facilitare la compilazione dei campi (figura 5.7).

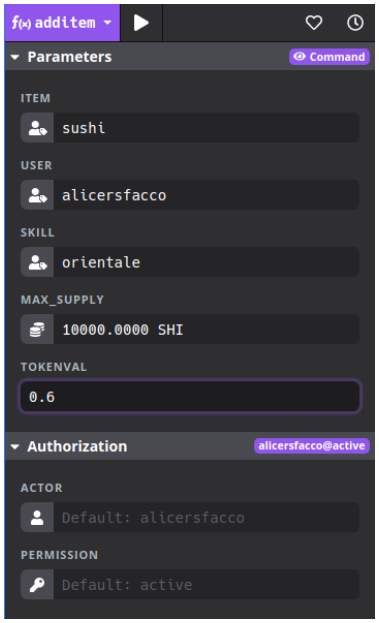


Figura 5.7: Compilazione dei campi dell'azione additem del contratto RatingSystem

Inoltre, tramite un API endpoint è possibile interagire con una testnet di EOS.IO direttamente da EOS Studio. Tramite l'applicativo si può effettuare qualsiasi operazione, tranne che la creazione degli account, che va fatta direttamente sul sito ufficiale della testnet generando le chiavi private e pubbliche per i permessi necessari ad autorizzare le azioni (figura 5.8).

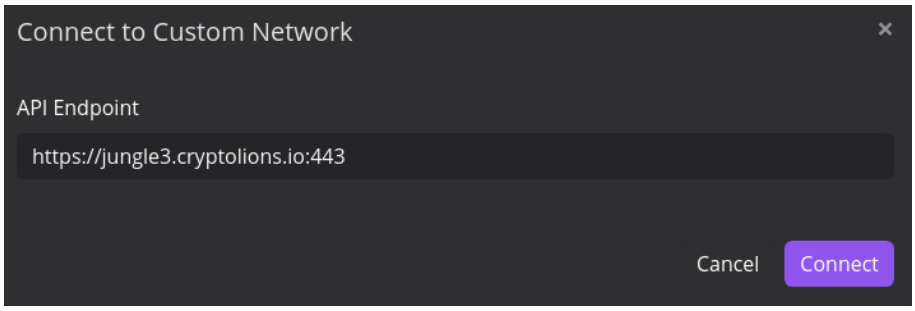


Figura 5.8: Connessione all'API endpoint della testnet Jungle3 per il monitoring

Capitolo 6

Valutazioni e discussione

In questo capitolo presenteremo tutti gli strumenti utilizzati per effettuare il testing dei contratti realizzati. Analizzeremo i costi di RAM, CPU e NET sia per il deploy dei contratti, sia per le azioni più rilevanti che sono state implementate. Si descriverà come sono stati effettuati i test e quali sono i costi che un ipotetico item owner dovrebbe sostenere utilizzando la piattaforma. Infine, verrà presentato un confronto con la precedente versione del Rating System implementata in Ethereum mettendo in risalto le particolarità di implementazione per spiegare le differenze che si presentano nei costi di deploy e delle operazioni.

Per effettuare il testing degli smart contract realizzati, sono state affrontate 3 fasi:

- scelta della testnet e delle piattaforme da utilizzare per l'analisi;
- valutazione sui costi di deploy del contratto e push delle azioni;
- valutazione sul comportamento dell'applicazione all'aumentare delle informazioni da salvare in termini dei costi delle risorse RAM, CPU, NET;
- confronto dei risultati ottenuti con la stessa DApp sviluppata su Ethereum.

6.1 Piattaforma e strumenti utilizzati per il testing

L'obiettivo principale nella fase di testing era utilizzare la testnet ufficiale di EOS.IO chiamata *Block.one*. Questa però non permetteva di visualizzare alcuni valori utili per effettuare l'analisi delle operazioni come ad esempio la RAM.

Da qui è scaturita la scelta di utilizzare una delle testnet più note: *Jungle* (figura 6.1). La particolarità di *Jungle* riguarda le prestazioni dimostrate in termini di performance per quanto riguarda le TPS (Transaction Per Second) raggiungendo un picco di 9000 [20].

Figura 6.1: Testnet Jungle3.0 utilizzata per il test della DApp

In combinazione, è stato utilizzato come servizio *blocks.io*, il quale permette di visualizzare varie informazioni relative sia alle mainnets, che alle testnet; nel nostro caso per *Jungle3.0*. Questo tipo di servizi viene chiamato *Blockchain Explorer*, cioè un servizio che permette di esplorare e visualizzare in tempo reale le informazioni che sono presenti sulla blockchain: come blocchi, transazioni, accounts e smart contract. Inoltre, permette di visualizzare tutte le transazioni effettuate da un account in particolare, filtrando per smart contract usato, oppure per azione. Infine, effettua un'analisi della risposta della

transazione data in json, fornendo informazioni relative all'uso di RAM, CPU e NET consumati durante l'esecuzione della transazione.

Per effettuare i test sul comportamento dell'applicazione è stata utilizzata la libreria di python *eospy* [8], la quale permette di effettuare i comandi di cleos e attraverso la risposta restituita in json, è stato possibile analizzare facilmente un numero consistente di transazioni effettuate.

Nella figura 6.2, è rappresentato il flusso di lavoro per arrivare all'analisi degli smart contract dei quali è stato fatto il deploy sulla blockchain. Inizialmente la DApp viene deployata sulla testnet *Jungle*. Il controllo delle singole transazioni viene effettuato tramite *blocks.io*. Infine, usando la libreria *eospy* è stato possibile scrivere un programma in python che effettua il *push* delle transazioni e ne riceve l'esito, il quale verrà analizzato prendendo solo le informazioni di interesse per conservarle in un file di *log*.

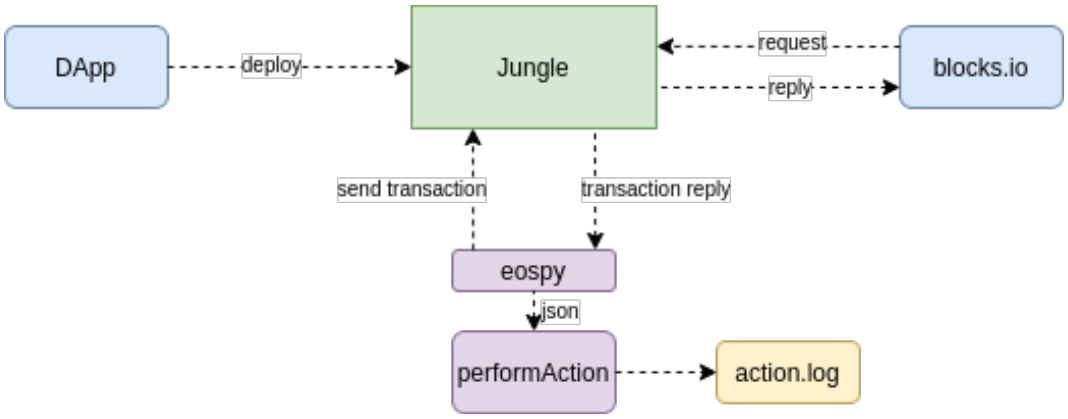


Figura 6.2: Diagramma dell'interazione delle componenti

6.2 Valutazione deploy Rating System

I costi relativi al deploy di un contratto, come anche quelli delle azioni, dipendono dal mercato e dal valore in termini di token di sistema (EOS) che è in continua variazione. Come è possibile vedere nel grafico (figura 6.3), infatti, il valore del token EOS cambia ripetutamente nelle 24 ore.

Figura 6.3: Grafico che mostra la variazione del valore di mercato di un EOS durante una sola giornata [2]

Oltre a queste variazioni, è necessario anche considerare i costi interni di EOS. Per costi interni ci si riferisce alle risorse che un account deve acquistare per il deploy di un contratto e per l'esecuzione delle transazioni. Il costo di RAM, CPU e NET non è unico, varia in base alla misura e all'importanza della risorsa.

I seguenti valori riportati, quindi, non rappresentano un prezzo fisso, ma servono per dare un'idea approssimata sui costi, ipotizzandone un possibile ammontare per il deploy della DApp [6].

- RAM = 0.03993292 EOS/KB;
- CPU = 3.67808520 EOS/ms/Day;
- NET = 0.00030145 EOS/KiB/Day.

Si noti la misura di CPU e NET, le quali, vengono ripristinate ogni tre giorni (2.4.3).

Ipotizziamo di dare un prezzo fisso anche per il token EOS considerando un costo medio degli ultimi mesi, ad esempio: 1 EOS = €2.57.

Analizzando i costi in tabella (figura 6.4), possiamo notare che le risorse RAM e NET non variano, poichè sono relative al contratto. Il valore della CPU d'altro canto varia pur facendo il deploy dello stesso contratto. Il valore rappresentato in figura è stato preso come costo medio in seguito a ripetuti deploy. Nella tabella (figura 6.4) sono riportati il numero di token di EOS necessari e il relativo equivalente in Euro.

Jungle3.0 - deploy

	ratingsystem	rsf.token
RAM (bytes)	587401	183775
CPU (µs)	1428	994
NET (bytes)	59960	19136
EOS	28,727	11,000
EURO	€ 73,83	€ 28,27

Figura 6.4: Tabella dei costi di deploy dei contratti

Definiamo come costi *istantanei*, i costi che vanno sostenuti al momento dell'esecuzione della transazione. Definiamo invece come *finali*, i costi che bisogna effettivamente sostenere a distanza di tre giorni dall'esecuzione.

Ai costi in tabella, bisogna sottrarre i costi di CPU e NET che dopo 3 giorni verranno riborsati. Il risultato è che nel costo di deploy istantaneo sono compresi CPU e NET, ma nel tempo i costi da mantenere, ovvero quelli finali, sono solo quelli di RAM (figura 6.5).

Costi Finali

	ratingsystem	rsf.token
EOS	28,727	11,000
- EOS rimborsati	-5,270	-3,662
EOS finali	23,457	7,339
EURO finali	€ 60,28	€ 18,86

Figura 6.5: Tabella dei costi finali di deploy dei contratti

6.3 Valutazione operazioni principali RatingSystem

Per quanto riguarda le operazioni, bisogna tener conto di quella che è la caratteristica principale degli smart contract in EOS: le tabelle multi-index con la possibilità di creare un database.

Dunque, l'unica risorsa sulla quale gravano i costi delle tabelle è la RAM, poichè le informazioni salvate occupano RAM. Possiamo riassumere i costi come:

- la quantità di informazioni da salvare in una riga;
- overhead per la creazione della tabella dal valore di 112 bytes;
- overhead per la creazione di una riga dal valore di 108 bytes;

Ovviamente, il costo di creazione della tabella si paga una sola volta, mentre l'overhead di riga lo si paga ad ogni azione che comporta l'aggiunta di una riga in una tabella.

Per questo, come mostra la figura 6.6, abbiamo nuovi campi da valutare. Quando un'azione viene eseguita per la prima volta creando una nuova tabella, allora il costo di RAM da pagare sarà più alto, inglobando il costo di overhaed. Quando la tabella esiste già, l'ammontare di RAM da pagare non tiene conto dell'overhead di tabella, ma solo dell'overhead di riga e della quantità di informazioni da salvare.

Jungle3.0 - operations

	User			Item owner			Administrator		
Operations	RAM (bytes)	CPU (µs)	NET (bytes)	RAM (bytes)	CPU (µs)	NET (bytes)	RAM (bytes)	CPU (µs)	NET (bytes)
addskill							120 [232]	260	104
newuser	121 [233]	291	104						
additem				913 [1137]	400	144			
payperm				409 [633]	497	136			
payitem	153 [409]	320	128	-153					
payitem + token	153 [281]	336	128	-153					
addrate	400 [1248]	325	120						
addrate + token	400 [400]	282	120	128					

Figura 6.6: Tabella dei costi delle operazioni

Alcune operazioni, che usano inline action, hanno dei costi aggiuntivi che non gravano sull'economia dell'utente che innesca l'azione. Un esempio di questa tipologia può essere l'operazione *addrate*, dove il proprietario dell'item ricompensa il cliente per la recensione inserita effettuando un trasferimento di token.

Altre invece, hanno un comportamento da "passaggio di responsabilità". Ad esempio, nell'operazione *payitem*, il cliente effettua il pagamento e di conseguenza si fa carico della riga che attesta il pagamento avvenuto. La riga veniva precedentemente creata dal proprietario dell'item tramite l'azione *payperm*. Ecco perchè viene liberata la RAM al proprietario dell'item che va a carico del cliente che paga.

Come è possibile notare, sempre in figura 6.6, distinguiamo le azioni payitem e addrate aggiungendo la *+ token*. Lo scopo è di diversificare le azioni in base al contesto in cui vengono eseguite. L'azione payitem può assumere comportamenti diversi se si paga in token RSF, si paga una parte del costo in coupon, si paga interamente in coupon. Se il pagamento viene fatto interamente in RSF o interamente in coupon, prendiamo come riferimento la normale payitem, dove la transazione prevede un unico trasferimento. Nel caso in cui il pagamento è diviso tra token RSF e coupon, allora nella transazione è previsto un doppio trasferimento, faremo quindi riferimento alla riga *payitem + token*.

L'azione addrate invece è diversificata per distinguere un'addrate senza ricompensa, nel caso in cui un utente abbia un valore di skill pari a 0 e non ottenga una retribuzione in coupon. Per la situazione appena citata faremo

riferimento alla normale addrate. Nel momento in cui un item owner deve ricompensare un cliente con dei coupon, faremo riferimento alla *addrate + token*.

Dopo aver appurato che non è possibile calcolare a priori la quantità di CPU che verrà utilizzata, rimane da capire se è possibile conoscere quale sarà il valore di NET. Sappiamo che la NET è calcolata in bytes, ogni byte viene visto come una parola, una networds.

Un'azione senza parametri è composta da 12 networds, ad ogni parametro che una transazione possiede si aggiunge una networds. Di conseguenza, prendendo in esempio l'azione *addskill*, che ha arietà uguale a uno, ci si aspetta di avere 13 networds, quindi

$$13 \text{ networds } * 8 \text{ bits } = 104 \text{ bytes}$$

Nella figura 6.7 sono invece rappresentati i costi in termini di EOS e convertiti in Euro. Come si può notare la tabella è divisa in due sezioni. Il secondo gruppo di colonne sta a indicare i costi finali, quando i costi di CPU e NET vengono rimborsati.

Jungle3.0 - costs of operations

	EOS [istantanei]	EURO [istantanei]	EOS [finali]	EURO [finali]
addskill	0,961	€ 2,47	0,005	€ 0,01
newuser	1,075	€ 2,76	0,005	€ 0,01
additem	1,508	€ 3,87	0,036	€ 0,09
payperm	1,844	€ 4,74	0,016	€ 0,04
payitem	1,183	€ 3,04	0,006	€ 0,02
payitem + token	1,242	€ 3,19	0,006	€ 0,02
addrate	1,211	€ 3,11	0,016	€ 0,04
addrate + token	1,053	€ 2,71	0,016	€ 0,04

Figura 6.7: Tabella dei costi delle operazioni in termini di EOS e di EURO

6.3.1 Valutazione funzioni di rating

Le funzioni che calcolano la media delle recensioni, al contrario delle funzioni viste in precedenza, non consumano alcuna RAM, poichè non vanno a

scrivere nuovi dati sulla blockchain o a modificare il suo stato. L'utente che eseguirà la transazione dovrà comunque pagare CPU e NET.

Sappiamo che il costo della CPU dipende dalla complessità del codice che, nel nostro caso, potrebbe dipendere dall'accesso alle tabelle.

Nella figura 6.8, sono rappresentate le relazioni tra alcune delle tabelle che fanno parte del database implementato dal modulo Rating System. Queste, sono le tabelle coinvolte nell'esecuzione della funzione di valutazione *weightedavg*.

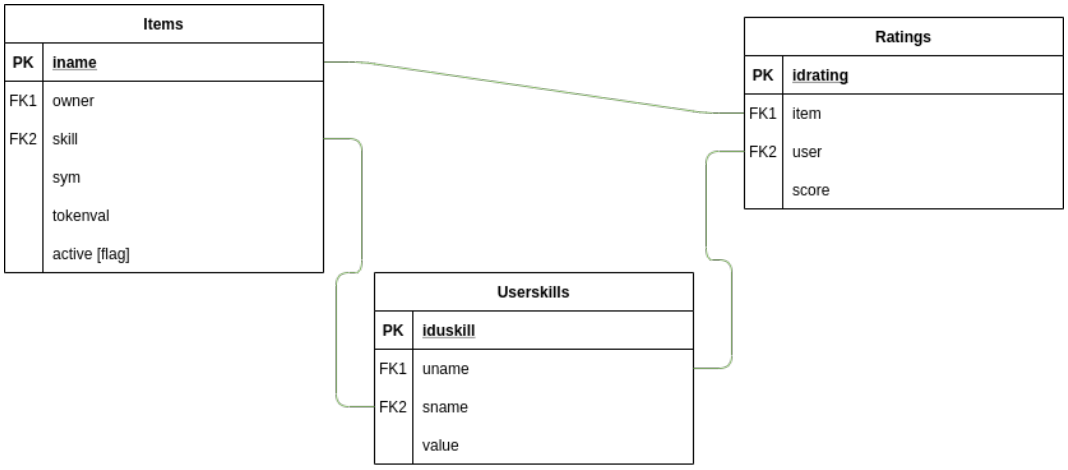


Figura 6.8: Relazioni tra le tabelle durante l'esecuzione della funzione weightedavg

Prendiamo in analisi la funzione *weightedavg*.

$$weightedavg = \frac{\sum_{i=1}^{n} r_i * p_i}{\sum_{i=1} p_i} \tag{6.1}$$

Dove nella funzione 6.1, r rappresenta il valore della recensione, e p è il peso che possiede quella recensione, dato dal valore della skill che un utente possiede.

Come si può vedere nella figura 6.8, una volta che si ha il valore della recensione, bisogna conoscere il punteggio della skill dell'utente che ha lasciato quel rate. Dunque, il valore della skill che si vuole conoscere è quello della skill che caratterizza l'item.

Gli accessi alla tabella *userskills* saranno molteplici, ma alcuni evitabili. Ad esempio, se un cliente ha lasciato n recensioni per lo stesso locale, si potrebbe evitare di accedere n volte alla tabella, ne basterebbe una. All'interno

della funzione *weightedavg*, viene utilizzato un vettore per salvare un utente e il suo punteggio di skill alla prima occorrenza, per evitare di accedere alla tabella altre volte. Tale ottimizzazione fatta al codice, permette, a chi fa uso dell'azione, di consumare meno CPU.

6.4 Test sulle azioni

Per confermare i dati raccolti, serviva sapere se il comportamento della DApp, in termini di costi nel tempo, rimanesse sempre lo stesso oppure se, in qualche modo, il costo della RAM potesse variare. In tal modo tutte le valutazioni fatte sui costi della RAM, dato che è l'unico costo che persiste nel tempo, sarebbero state inutili.

L'idea è stata quella di effettuare numerose transazioni utilizzando due account per fare questo tipo di test. Come fatto precedentemente, utilizzeremo Alice e Bob come attori. Alice ha inserito il suo item nel sistema, mentre Bob è un normale cliente. Seguendo la logica dell'applicazione:

- 2000 payperm;
- 1000 payitem pagando in RSF;
- 1000 payitem pagando con il token/coupon;
- 2000 addrate (si ricorda che dopo la prima transazione diventa una addrate con trasferimento di token).

Il comportamento precedentemente descritto è implementato in python, mediante la libreria *eospy*. La struttura del codice è simile per ogni azione:

1. connessione alla testnet tramite l'API usando la funzione della libreria che si interfaccia con cleos;
2. un ciclo for per effettuare x transazioni;
3. descrizione degli argomenti dell'azione, specificare il nome dell'azione che si vuole effettuare accompagnato dal nome dello smart contract dove risiede il codice della transazione, indicare l'autorizzazione (nome dell'account EOS.IO e il livello di permesso da utilizzare);

```
jungle = eospy.cleos.Cleos(url=
    'https://jungle3.cryptolions.io:443')

def payperm(num_action):
    bill=0.50
    start = time.time()
    resp = []
    for i in range(num_action):
        bill = ((bill + 0.5)%1)+1
        arguments = {
         "item": "sushi",
         "owner": "alicersfacco",
         "client": "bobrsfaccoun",
         "bill": "{price:.4f} RSF".format(price=bill)
        }
        payload = {
         "account": "ratingsystem",
         "name": "payperm",
         "authorization": [{
           "actor": "alicersfacco",
           "permission": "active",
         }],
        }
```

Codice 6.1: Preparazione dei dati della transazione payperm, fasi 1-2-3

4. convertire tutte le informazioni in binario;
5. formare la transazione finale inserendo anche la data e ora attuale;
6. inserire la chiave privata relativa al permesso utilizzato;

```
#Converting payload to binary
data = jungle.abi_json_to_bin(payload['account'],
    payload['name'], arguments)
#Inserting payload binary form as "data" field
#in original payload
payload['data'] = data['binargs']
#final transaction formed
trx = {"actions": [payload]}
trx['expiration'] = str((datetime.utcnow() +
  datetime.timedelta(seconds=60)).replace(tzinfo=pytz.UTC))

key = eospy.keys.EOSKey(accountkey)

resp.append(jungle.push_transaction(trx,
    key, broadcast=True))
```

Codice 6.2: Fase di preparazione dei dati della transazione peyperm, fasi 4-5-6

7. effettuare il push della transazione;
8. analisi della risposta in json ottenuta con l'estrapolazione delle seguenti informazioni:
   - RAM consumata oppure rimborsata a seconda della logica dell'azione usata;
   - il numero del primo e dell'ultimo blocco nei quali sono state salvate le transazioni;
   - la quantità di blocchi trascorsi dalla prima all'ultima transazione;
   - il numero di blocchi diversi nei quali sono state inserite le transazioni;
9. le precedenti informazioni raccolte insieme al valore del tempo, espresso in secondi, trascorso dalla prima all'ultima transazione effettuata, vengono scritte su un file di *log*.

```
with open("actions.log", "a") as log:
    if (action == "payitem" and pay_with_token):
        log.writelines(["##### ", str(numAction) , " " ,
            action , " with coupon performed ##### \n"])
    else:
        log.writelines(["##### ", str(numAction) , " " ,
            action , " performed ##### \n"])
    for act in actionList:
        blocksNum.append(act['processed']['block_num'])
        delta_ram = act['processed']['action_traces']
            [0]['account_ram_deltas']
        for delta in delta_ram:
            log.writelines([delta['account'], ": " ,
                "{}".format(delta['delta']), "\t" ])
            log.write("\n")
    blocksElapsed = blocksNum[len(blocksNum) - 1]
        - blocksNum[0]
    log.writelines(["first block: ",
        str(blocksNum[0]) , " - last block: ",
        str(blocksNum[len(blocksNum) - 1]) ,"\n"])
    log.writelines(["blocks elapsed: ",
        str(blocksElapsed) , "\n"])
    #no duplicates in list
    #to achieve the number of different blocks
    blocksNum = list(dict.fromkeys(blocksNum))
    log.writelines(["number of different blocks: ",
        str(len(blocksNum)) , "\n"])
    log.writelines(["time elapsed: ",
        str(performedAction[1]) , "\n"])
    log.write("############################## \n")
```

Codice 6.3: Push della transazione payperm e analisi della risposta in json, fasi 7-8-9

Si noti che la testnet riconosce la stessa azione ripetuta nell'arco di un tempo ristretto, bloccando così l'esecuzione delle successive per motivi di sicurezza. Per risolvere questo "problema" viene fatta una semplice operazione per modificare ad ogni iterazione il prezzo che il cliente dovrà pagare (6.1).

Il risultato del file di log avrà una struttura simile al codice 6.4, il quale rappresenta l'esecuzione di 5 payitem. Si può anche notare che il costo delle operazioni rimane sempre costante.

```
alicersfacco: -153  bobrsfaccoun: 153
alicersfacco: -153  bobrsfaccoun: 153
alicersfacco: -153  bobrsfaccoun: 153
alicersfacco: -153  bobrsfaccoun: 153
alicersfacco: -153  bobrsfaccoun: 153
first block: 57432125 - last block: 57432131
block elapsed: 6
number of different blocks: 5
time elapsed: 3.876619577407837
```

Codice 6.4: Risultato del file di log

6.5 Calcolo token in stake

Vogliamo ora anaòizzare quanto costa sostenere un account possessore di un item su ESO.IO facendo uso della nostra DApp.
Immaginiamo di dover analizzare l'account di un cliente che ha inserito il suo ristorante e utilizza lo smart contract unicamente per dare i permessi di pagamento ai clienti.

Abbiamo già detto che CPU e NET si ripristinano in 3 giorni. Quindi, il nostro obiettivo è capire quanti token l'account deve tenere in stake per essere libero di effettuare le sue operazioni.

Ipotizziamo che l'utente Alice, che è proprietario di un item, in media giornalmente effettui 30 permessi di pagamento. Gli utenti a cui è destinato il permesso di pagamento lo eseguono nello stesso giorno, in più aggiungono anche una recensione.

Sappiamo già che una payitem costa 1,844 EOS (figura 6.7). A questo punto abbiamo 30 *payperm* al giorno per 3 giorni, per un numero di token pari a 165,96 EOS, equivalenti a 426,51 EURO.

Questo valore calcolato rappresenta il costo in 3 giorni solo dell'azione payperm. Allo scadere dei 3 giorni le spese di CPU e NET vengono "riborsate", quindi al termine dei 3 giorni l'unico costo che ci resta da sostenere è la RAM, per un valore in EOS pari a 1,467 equivalenti a 3,77 EURO.

Sappiamo che il cliente, quando effettua il pagamento si assume la responsabilità della riga per il pagamento. Indipendentemente se paga o meno con i token/coupon ad Alice si libereranno 153 bytes di RAM.

Supponiamo ora che tutti i clienti abbiano già una skill avanzata, quindi Alice dovrà ricompensare la loro recensione con i suoi token. Per effettuare questa transazione, consumerà 128 bytes di RAM. In totale, la RAM che Alice dovrà mantenere sarà di 384 byte. Considerando tutte le 90 operazioni avrà un costo di 1,377 EOS equivalenti a 3,53 EURO.

Quindi possiamo dedurre che al termine dei tre giorni l'unico costo del quale bisogna tenere conto, risulta equivalente in EOS di 384 bytes di memoria. Infine, oltre alla RAM utilizzata da tutte le azioni, teniamo conto sia di CPU e NET, per un totale di 164,16 EOS equivalenti a 421,89 EOS.

Concludiamo dicendo che la quantità necessaria di EOS che Alice dovrà tenere nel suo account, per eseguire le varie operazioni nei 3 giorni, sarà di 165,537 EOS. Senza dimenticare di dover sostenere periodicamente costi per l'acquisto della RAM sulla quale salvare le transazioni da effettuare.

6.6 Confronto con la versione in Ethereum

Per fare un confronto abbiamo bisogno di spiegare alcune particolarità dell'implementazione fatta in Ethereum. Il confronto che effettueremo considererà da una parte l'implementazione fatta su EOS.IO e dall'altra avremo il sistema sviluppato e valutato su Ethereum in [30]. La struttura dell'applicazione è divisa in più smart contract dove ognuno rappresenta una classe (figura 6.9). Vi sono più smart contract per gestire separatamente le operazioni. Abbiamo contratti per gestire le operazioni di: sistema, utenti, item, skill e funzioni di valutazione.

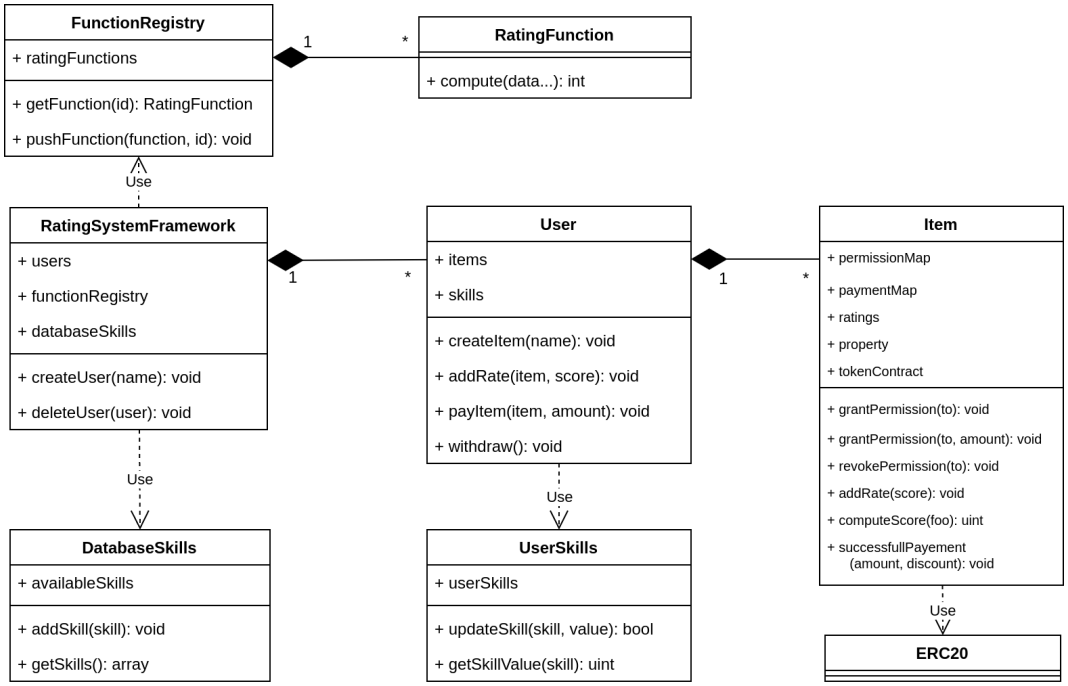


Figura 6.9: Divisione dei moduli di Ethereum

Facendo riferimento alla figura 6.9, in Ethereum, non viene fatto il deploy di tutto il contratto direttamente. L'implementazione adottata, consiste nel fare il deploy di un contratto nel momento in cui viene richiesto. Ad esempio, come si può vedere in figura 6.10 l'operazione *createUser* del contratto RatingSystemFramework ha il compito di fare il deploy del contratto User nel momento in cui un nuovo utente entra a far parte del sistema. Allo stesso

modo, quando un utente vuole aggiungere il suo item, attraverso l'operazione *createItem*, viene effettuato il deploy del contratto Item.

Quindi, a parità di applicazione, in EOS è stato sviluppato un solo contratto più uno per la gestione dei token; mentre in Ethereum ben sette contratti più uno per la gestione dei token.

Contratto	Operazione	Costo (gas)	Eth-15Gwei
RatingSystemFramework	deploy	7 027 645	0.1054
RatingSystemFramework	createUser	4 201 598	0.0630
User	createItem	1 998 003	0.0299
User	payItem	66 898	0.0011
User	addRate	347 004	0.0052
Item	grantPermission	79 034	0.0011
DatabaseSkills	addSkill	68 992	0.0010
SimpleAverageComputer	deploy	219 359	0.0032
WeightedAverageSkillComputer	deploy	242 438	0.0036
ComputerRegistry	pushComputer	75 637	0.0011

Figura 6.10: Costo in gas e in Eth considerando un prezzo del gas pari a 15 Gwei

Come possiamo notare (figura 6.10), chi è amministratore del Rating System in Ethereum, fa il deploy solo di tre contratti. Quando viene effettuato il deploy del RatingSystemFramework, al suo interno è compreso anche il deploy del contratto DatabaseSkills. Ad oggi effettuare il deploy di questo contratto costa 160.65 EURO, mentre in EOS (figura 6.4) ha un costo di 73.83 EURO istantanei e di 60.28 EURO finali.

Un utente in EOS.IO, però, paga solo l'esecuzione delle transazioni assumendosi i costi a tempo indeterminato della RAM. Mentre invece, in Ethereum un utente deve pagare anche per il deploy di un contratto, costo che in EOS è destinato solo all'account amministratore del Rating System.

Mettendo a paragone le operazioni di EOS che su Ethereum non comportano il deploy di un contratto, facciamo riferimento alla figura 6.11. Come abbiamo già visto, in EOS, ci sono costi istantanei e costi finali. Dalla tabella di confronto si può vedere come alcuni costi risultino più consistenti rispetto ad altri. La spiegazione sta in come vengono gestite le operazioni. Come annunciato prima, le operazioni quali createUser e addItem effettuano il deploy di un contratto, di conseguenza hanno un costo maggiore rispetto a quelle che sono

semplicemente operazioni di un contratto. Senza dimenticare che, effettuare il deploy di un contratto Users, in Ethereum, comporta il deploy del contratto UserSkills, il quale implementa una struttura dati per tener conto delle skill che l'utente acquisisce interagendo con il sistema.

Inoltre, valutando quella che è una particolarità di EOS, il rimborso delle risorse CPU e NET, notiamo che alcune operazioni hanno un costo istantaneo che è maggiore rispetto al costo che si ha in Ethereum. Però, sebbene i costi iniziali siano maggiori, il costo viene drasticamente ammortizzato dopo i tre giorni.

	Ethereum		EOS.IO		
Operations [nome Ethereum/EOS]	Eth-15Gwei	EURO	EOS	EURO [istantanei/finali]	
addSkill/addskill	0,0010	€ 1,57	0,961	€ 2,47	€ 0,01
createUser/newuser	0,0630	€ 100,73	1,075	€ 2,76	€ 0,01
additem/createItem	0,0299	€ 47,95	1,508	€ 3,87	€ 0,09
grantPermission/payperm	0,0011	€ 1,80	1,844	€ 4,74	€ 0,04
payItem/payitem	0,0011	€ 1,52	1,183	€ 3,04	€ 0,02
addRate/addrate	0,0052	€ 7,89	1,053	€ 2,71	€ 0,04

Figura 6.11: Confronto costi Ethereum vs EOS

Capitolo 7

Conclusioni e Sviluppi futuri

Lo scopo principale di questo tirocinio è stato quello di implementare il sistema di recensione "Rating System" precedentemente implementato in Ethereum [36].

Nella prima parte del tirocinio sono state messe in evidenza le caratteristiche e le potenzialità della blockchain EOS.IO quali l'algoritmo di consenso, le performance che ne derivano, la gestione dei permessi e delle risorse degli account. È seguito lo studio dell'algoritmo di consenso "Delegated Proof of Stake" e delle modalità con cui viene applicato dalla blockchain. Successivamente, sono state confrontate le blockchain Ethereum ed EOS.IO secondo quelle che vengono individuate come caratteristiche da valutare per evidenziare le differenze tecniche che le distinguono.

Nel Rating System implementato, un utente si registra sul sistema, può decidere di parteciparvi da cliente oppure da possessore di un item. Un item owner concede i pagamenti ai clienti, un cliente effettua il pagamento e da quel momento in poi ha la possibilità di inserire nel sistema una recensione relativa alla sua esperienza con il servizio ricevuto. Quando un cliente inserisce la sua recensione, l'item owner lo ricompensa con dei token. Lo scopo è che l'utente possa utilizzare questi token ricevuti per effettuare i prossimi pagamenti allo stesso item usandoli come coupon.

Successivamente, lo stesso sistema è stato implementato in EOS.IO sfruttando le caratteristiche che vengono proposte negli smart contract della bloc-

kchain EOS.IO, tra tutte di sicuro ricordiamo la possibilità di creare un vero e proprio database interno.

Grazie ai vari tool utilizzati e alla testnet Jungle, sulla quale è stato fatto il deploy dei contratti, è stato possibile valutare le transazioni effettuate e raccogliere i dati per poi passare alla fase di analisi.

I risultati ottenuti sono stati messi a confronto con la precendente implementazione di Ethereum. Abbiamo visto come sono gestite nelle transazioni le risorse degli account e di conseguenza, come vengono calcolati i costi del deploy dei contratti e dell'esecuzione delle operazioni in essi contenute.

Quello che risulta dal confronto è che le diverse piattaforme danno spazio a diverse implementazioni. Abbiamo individuato molti vantaggi nella blockchain EOS.IO a partire dalle alte performance, ovvero transazioni confermate in tempo minore rispetto a Ethereum, fino ad arrivare ai costi di gestione e costi per l'esecuzione delle transazioni più ammortizzati. Inoltre gli smart contract in EOS.IO danno l'opportunità di creare al proprio interno un vero e proprio database, utile per gestire e organizzare le informazioni.

Valutando i dati ottenuti, possiamo dire che, utilizzare l'implementazione del rating system su EOS.IO è vantaggioso per l'utente non solo per il meccanismo di tokenizzazione, ma anche per i costi per l'esecuzione delle transazioni.

7.1 Sviluppi futuri

Abbiamo descritto ed esaminato molte delle potenzialità di EOS.IO a partire dall'uso delle tabelle come database, fino a godere dell'alta scalabilità della blockchain. Abbiamo ampiamente parlato della possibilità di creare dei sottogruppi sfruttando la gestione della struttura gerarchica dei permessi che EOS mette a disposizione. Immaginiamo di poter sfruttare questa caratteristica per il Rating System. Una delle possibili alternative di implementazione potrebbe consolidarsi nella creazione di un gruppo "RatingSystem" da parte del gestore.

Lo scopo potrebbe essere quello di aggiungere al gruppo direttamente l'account che si registra nel sistema. In tal modo, non sarebbe più necessario avere una tabella users nel database e gestire in modo diverso la logica del modulo

del sistema di recensione. Senza la tabella, non sarebbe più necessario pagare un overhead per l'aggiunta della riga e ridurre i costi per entrare a far parte del sistema.

Dal punto di vista della sicurezza, potrebbe essere necessario modificare il codice di alcune transazioni del sistema aggiungendo la possibilità di avere azioni *multisignature*. Questa implementazione potrebbe risultare vantaggiosa nel momento in cui per l'esecuzione di una transazione venga richiesta l'autorizzazione dell'utente e del gestore del Rating System, ad esempio nell'azione *additem*. Attualmente, chiunque può creare un token nel sistema, utilizzando l'azione create token. Questa scelta implementativa è dovuta al fatto che la creazione della riga nella tabella deve essere attribuita all'utente che sta aggiungendo un item al sistema. Quindi è necessariamente richiesta l'autorizzazione dell'utente e non del Rating System, altrimenti dovrebbe essere il gestore del sistema a pagare per la creazione della riga. Questo problema non è risolvibile facendo uso del solo gruppo di autorizzazioni. È necessario che nella creazione dell'item, ed in particolare nella transazione *create*, siano richieste entrambe le autorizzazioni dell'utente e del gestore del sistema.

Bibliografia

[1] Coin Market. https://bit.ly/3aTcwMo.

[2] CoinGecko. https://bit.ly/2ZZ8Nb5.

[3] Cos'è un Algoritmo di Consenso Blockchain? https://bit.ly/3cJ5WKR.

[4] Cos'è un'ICO (Initial Coin Offering) e Come Funziona? https://bit.ly/2Nejtzk.

[5] EOS Mainnet. https://bit.ly/3pdV2PV.

[6] EOS Nation. https://erp.eosnation.io/.

[7] EOS Studio IDE. https://bit.ly/3pdgcxD.

[8] eospy. https://bit.ly/2N3khHx.

[9] Ethereum Developers. https://bit.ly/3rsyQmA.

[10] EOS.IO technical white paper v2. https://bit.ly/3j3ge9K, 2018.

[11] Introduction to EOS.IO. https://bit.ly/3tejwM8, 2018. Developer Portal.

[12] Irreversible block in EOS.IO. https://bit.ly/3r9m2Bq, 2018. Developer Portal, Glossary section.

[13] How we Calculate EOS Resource Usage. https://bit.ly/3rKUNgE, 2019.

[14] Introduction to EOS. https://bit.ly/3uOardR, 2019.

[15] Multisignature Accounts On EOS: How Do They Work? `https://bit.ly/2PgtFIx`, 2019.

[16] What's the Maximum Ethereum Block Size? `https://bit.ly/2MOI1PL`, 2019.

[17] Decentralized Finance - What is DeFi in Cryptocurrency? `https://bit.ly/3q5H4A2`, 2021.

[18] Hard Fork. `https://bit.ly/3a9z64a`, 2021.

[19] Alfredo de Candia. Come funziona il modello di consenso di EOS.IO. `https://bit.ly/2N2Ludb`, 2020.

[20] Alfredo de Candia. EOS in testa per TPS grazie alla nuova testnet. `https://bit.ly/3tw8bHf`, 2020.

[21] Filippo Angeloni. Diversi tipi di consenso nella Blockchain. `https://bit.ly/304aLqk`, 2019.

[22] Michael Barborak, Anton Dahbura, and Miroslaw Malek. The consensus problem in fault-tolerant computing. *ACM Computing Surveys (CSur)*, 25(2):171–220, 1993.

[23] Matthew Beedham. Here's the difference between 'permissioned' and 'permissionless' blockchains. `https://bit.ly/36PtLgq`, 2018.

[24] Belavadi Prahalad. Merkle proofs Explained. `https://bit.ly/3tiwb0A`, 2018.

[25] A. Bernasconi, P. Ferragina, and F. Luccio. *Elementi di crittografia*. Manuali / Pisa university press. Pisa University Press, 2015.

[26] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[27] Daniel Larimer, et al. EOS.IO white papers & blockchain philosophy. `https://bit.ly/3jeiuLq`, 2017.

[28] Simply Explained. Proof-of-Stake (vs proof-of-work). `https://bit.ly/3dXjJhi`, 2018.

[29] Eyal Hertzog and Guy Benartzi and Galia Benartzi. Bancor Protocol Continuous Liquidity for Cryptographic Tokens through their Smart Contracts, March 2018.

[30] Samuel Fabrizi. Strategia di tokenizzazione di un sistema di recensione su blockchain. pages 70–75, 2019.

[31] Fama. What are Smart Contract? `https://bit.ly/37XB5qR`, 2020.

[32] Martin Garriga, Stefano Dalla Palma, Maxmiliano Arias, Alan De Renzis, Remo Pareschi, and Damian Andrew Tamburri. Blockchain and cryptocurrencies: A classification and comparison of architecture drivers. *Concurrency and Computation: Practice and Experience*, page 5992, 2020.

[33] Giovanni Capaccioli. Una variante della classica Proof of Work: Ethereum. `https://bit.ly/3cL3Z0j`, 2019.

[34] Patrick Goh. 5 benefits of using a smart contract and how you can use them too. `https://bit.ly/3b1HfIj`, 2018.

[35] Kirill Shilov. Should Smart Contracts be Non-Turing Complete? `https://bit.ly/3oVzCa2`, 2019.

[36] Andrea Lisi, Andrea De Salve, Paolo Mori, and Laura Ricci. A smart contract based recommender system. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 29–42. Springer, 2019.

[37] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery.

[38] Loi Luu, Yaron Velner, Jason Teutsch, and Prateek Saxena. Smartpool: Practical decentralized pooled mining. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1409–1426, 2017.

[39] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. Blockchain based access control. In *IFIP international conference on distributed applications and interoperable systems*, pages 206–220. Springer, 2017.

[40] Marco Fucito. Ethereum: Transazioni che restano in attesa per lungo tempo. `https://bit.ly/39Czqb8`, 2017.

[41] Matteo Leibowitz. EOS: Don't Believe The Hype. `https://bit.ly/39STzKe`, 2018.

[42] Matteo Mordacchini, Ranieri Baraglia, Patrizio Dazzi, and Laura Ricci. A p2p recommender system based on gossip overlays (prego). In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 83–90. IEEE, 2010.

[43] Leandro Nascimento. Blockchain in a nutshell. `https://bit.ly/2NOqsj8`, 2018.

[44] Julien Noizet. Blockchain everywhere. `https://bit.ly/2OcinV6`, 2015.

[45] BitSongOfficial Decentralized Music Streaming Platform. Introduction to Staking. `https://bit.ly/3dXpgo4`, 2019.

[46] Michele Porta. DApp: cosa sono e come funzionano le applicazioni decentralizzate. `https://bit.ly/3pETXBI`, 2019. DApps in blockchain.

[47] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to Recommender Systems Handbook*, pages 1–35. Springer US, Boston, MA, 2011.

[48] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2016.

[49] Denish Shah and Emily Shay. *How and Why Artificial Intelligence, Mixed Reality and Blockchain Technologies Will Change Marketing We Know Today.* 02 2019.

[50] Bruno Skvorc. Ethereum: How Transaction Costs are Calculated. `https://bit.ly/3jpmmsZ`, 2018.

[51] Nick Vogel. The great decentralization: how web 3.0 will weaken copyrights. *John Marshall Review of Intellectual Property Law*, 15(1), 2016.

[52] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In Jan Camenisch and Doğan Kesdoğan, editors, *Open Problems in Network Security*, pages 112–125, Cham, 2016. Springer International Publishing.

[53] K. Wüst and A. Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54, 2018.

[54] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.