UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

Sviluppo e validazione di strumenti open source per protocolli di certificazione e tracciabilità basati su DLT

Development and Validation of Open-Source Tools for Certification and Traceability Protocols based on DLT

Candidato:
Marco Farinasso

Relatore:
Prof. Marco Baldi

Correlatore:
Prof. Paolo Santini

Anno Accademico 2022-2023

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

Sviluppo e validazione di strumenti open source per protocolli di certificazione e tracciabilità basati su DLT

Development and Validation of Open-Source Tools for Certification and Traceability Protocols based on DLT

Candidato:
Marco Farinasso

Relatore:
Prof. Marco Baldi

Correlatore:
Prof. Paolo Santini

Anno Accademico 2022-2023

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Alle mie dolci nipotine Ariel e Diana

Abstract

In this thesis, the development of a certification and traceability framework based on Distributed Ledger Technology (DLT) is proposed. Initially, an overview of the fundamental concepts of DLT is provided, focusing on distinctive features such as decentralization and data immutability. The specific topic of blockchain is then explored, including its general structure and division into public, private, and authorized blockchains.

Subsequently, a detailed examination of the Ethereum network is conducted, starting from transaction structure to concepts of tokens and smart contracts. The work continues with a thorough description of the state of the art of certification and traceability protocols utilizing blockchain technology, often combined with technological solutions such as RFID and NFC sensors and decentralized storage systems like IPFS.

Comprehensive details are then provided regarding the technologies used in the proposed framework, including programming language, libraries, database, and the simulation environment for interaction with the Ethereum network. The culmination of the thesis work is the introduction of the different proposed architectures. In the former approach, each piece of data is individually certified through an Ethereum transaction while, in the latter, data to be certified is organized in a data structure called a Merkle tree, and only its root is inserted into a transaction.

It is crucial to emphasize that the concept of the Merkle tree is the core of the work. Due to its characteristics, certification costs can be significantly reduced, and the verification phase is simplified through the concept of Merkle proof.

The thesis provides an in-depth analysis of the simulation results for various approaches, highlighting their strengths and weaknesses in terms of costs and performance based on the volume of data to be certified and the number of Merkle trees created.

Then, a cost function is presented, taking into account factors such as Ethereum cost, storage, and verification time, to generalize the treatment and make it applicable to different use cases. Finally, in the concluding part of the thesis, practical scenarios of interest are outlined, identifying optimal solutions by minimizing the cost modeling function.

Sommario

Nella tesi, si propone lo sviluppo di uno schema di certificazione e tracciabilità basato su Distributed Ledger Technology (DLT). Inizialmente, si offre una panoramica dei concetti fondamentali della DLT, concentrandosi sulle caratteristiche peculiari che la contraddistinguono quali la decentralizzazione e l'immutabilità dei dati, per poi approfondire il tema specifico della blockchain, di cui viene presentata la struttura generale e la divisione in blockchain pubblica, blockchain privata e blockchain autorizzata.

Successivamente, si esamina con dovizia di particolari la rete Ethereum, partendo dalla struttura delle transazioni fino ai concetti di token e di smart contract. Il lavoro prosegue con una minuziosa descrizione dello stato dell'arte dei protocolli di certificazione e tracciabilità che ricorrono alla tecnologia blockchain, la quale viene spesso combinata con soluzioni tecnologiche quali sensori RFID e NFC e sistemi di archiviazione decentralizzati quale IPFS.

Vengono poi forniti dettagli approfonditi riguardanti le tecnologie impiegate nello schema proposto come il linguaggio di programmazione e le librerie utilizzate, il database e l'ambiente di simulazione per l'interazione con la rete Ethereum. Il culmine del lavoro di tesi è rappresentato dall'introduzione delle diverse architetture proposte: nella prima ogni dato viene certificato singolarmente attraverso una transazione Ethereum mentre nel secondo i dati da certificare vengono organizzati in una struttura dati chiamata Merkle tree, di cui la sola radice viene inserita in una transazione.

È cruciale sottolineare che il concetto di Merkle tree è il cuore pulsante del lavoro, in virtù delle sue caratteristiche si riuscirà a ridurre in modo significativo i costi di certificazione e si semplificherà la fase di verifica grazie al concetto di Merkle proof.

La tesi offre, pertanto, un'analisi approfondita sui risultati della simulazione dei vari approcci mettendo in mostra i pregi e i difetti che li contraddistinguono in termini di costi e prestazioni al variare della numerosità dei dati da certificare e dal numero di Merkle tree che si creano.

In seguito, viene presentata una funzione costo che tiene conto di molteplici fattori quali il costo in ethereum, lo storage e il tempo di verifica in modo da poter generalizzare il più possibile la trattazione e renderla applicabile a diversi esempi applicativi. Infine, nella parte finale della tesi, sono delineati alcuni scenari di interesse pratico per cui si andrà ad individuare la soluzione ottimale minimizzando la funzione che modella i costi.

Indice

Introduzione **1**

1 Distributed Ledger Technology **3**
1.1 Blockchain . . . 4
1.1.1 Bitcoin . . . 5
1.2 Ethereum . . . 6
1.2.1 Il funzionamento della rete Ethereum . . . 6
1.2.2 Smart contract . . . 8
1.2.3 I Token Ethereum . . . 9

2 Studio dello stato dell'arte **11**
2.1 Tracciabilità . . . 11
2.1.1 Food Trust . . . 11
2.1.2 Tracciabilità agricola per HACCP . . . 12
2.1.3 Agri-BlockIoT . . . 14
2.1.4 Blockchain for food tracking . . . 15
2.2 Certificazione . . . 17
2.2.1 BlockIPFS . . . 17
2.2.2 Blockcert . . . 19
2.2.3 CertChain . . . 20
2.2.4 Smart contracts per certificati scolastici . . . 21
2.2.5 Schema basato su multi-firma . . . 22

3 Tecnologie e strumenti utilizzati **25**
3.1 JavaScript . . . 25
3.1.1 Node.js . . . 26
3.2 MongoDB . . . 26
3.2.1 MongoDB Compass . . . 27
3.3 Ganache . . . 27

4 Architettura basata su transazioni singole **29**
4.1 Funzione hash . . . 29
4.2 Organizzazione del Database . . . 32
4.3 Certificazione basata su singole Transazioni . . . 40
4.3.1 Verifica della certificazione . . . 47

5 Architettura basata su Merkle tree **53**
5.1 Merkle tree 53
5.2 Certificazione basata su Merkle tree 54
5.2.1 Verifica della certificazione 59

6 Analisi delle prestazioni e criteri di progetto **63**
6.1 Confronto delle prestazioni e i costi 63
6.2 Generalizzazione della funzione costo e applicazioni 68

7 Conclusioni **71**

Elenco delle figure

1 Schema dell'applicazione. 1

1.1 Architetture di un sistema centralizzato e distribuito [3]. 4
1.2 Struttura dati di una blockchain [4]. 4
1.3 Logo di Bitcoin [6]. 5

2.1 Framework del sistema di tracciabilità [17]. 13
2.2 Possibili utilizzi: da schemi centralizzati a decentralizzati [17]. 13
2.3 Architettura Agri-BlockIoT [19] 14
2.4 Architettura basata su blockchain per il tracciamento alimentare [13]. 16
2.5 Architettura di sistemi di tracciamento alimentare convenzionale [13]. 16
2.6 Sistema proposto [13]. 17
2.7 Confronto Ethereum e Hyperledger [13]. 17
2.8 Architettura BlockIPFS [20]. 19
2.9 Architettura Blockcerts [21]. 20
2.10 Control Service UML [23]. 20
2.11 Architettura certchain [23]. 21
2.12 Modello dei dati [23]. 22
2.13 Architettura di sistema [25]. 22
2.14 Architettura di MongoDB [25]. 23
2.15 Workflow [25] 24

3.1 logo JavaScript [26]. 25
3.2 Ganache 28

4.1 Funzione suriettiva [34]. 29
4.2 Caratteristiche delle funzioni hash note [36]. 30
4.3 Schema Merkle Damgård [37]. 31
4.4 Sponge construction [38]. 31
4.5 Documento informativo sulle singole aziende 32
4.6 Documento sui contenuti 33
4.7 Form per il caricamento dei dati 34
4.8 Andamento (in €) del prezzo dell'Ether a settembre 2023 [39]. 44
4.9 Output terminale 47
4.10 Output della Verifica 51

5.1 Merkle tree [40]. 53

5.2 Output della fase di certificazione 59
5.3 Verifica della certificazione con Merkle tree 62

6.1 Grafico di confronto per CPU usage 65
6.2 Grafico di confronto per lo storage 65
6.3 Grafico di confronto per Execution Time 66
6.4 Grafico di confronto per Transactions Cost 66
6.5 Grafico di confronto per Execution Time nella verifica 68

Introduzione

La crescente digitalizzazione dei processi produttivi in molteplici settori ha generato un'enorme quantità di dati digitali, andando così a rappresentare un valore essenziale per diverse organizzazioni e istituzioni. Tuttavia, garantire l'integrità e la certificazione di tali dati è diventata una sfida cruciale nell'era digitale. La manipolazione non autorizzata dei dati potrebbe compromettere l'affidabilità delle informazioni e minare la fiducia dei fruitori. Gli scenari di utilizzo possono essere i più disparati, a partire dalla certificazione di processi trasformativi volti, per esempio, a ridurre lo spreco di cibo fino alla proof of attendance, ovvero la prova di partecipazione ad un evento.

Questo lavoro di tesi mira a studiare il problema della certificazione e della tracciabilità dei dati, individuando un'architettura che sia in grado di sfruttare i vantaggi intrinseci della tecnologia a ledger distribuito (DLT). Nell'applicazione che si andrà a realizzare, i dati da certificare possono provenire da dispositivi embedded, mobile app o interfaccia web, come mostrato in Figura 1.

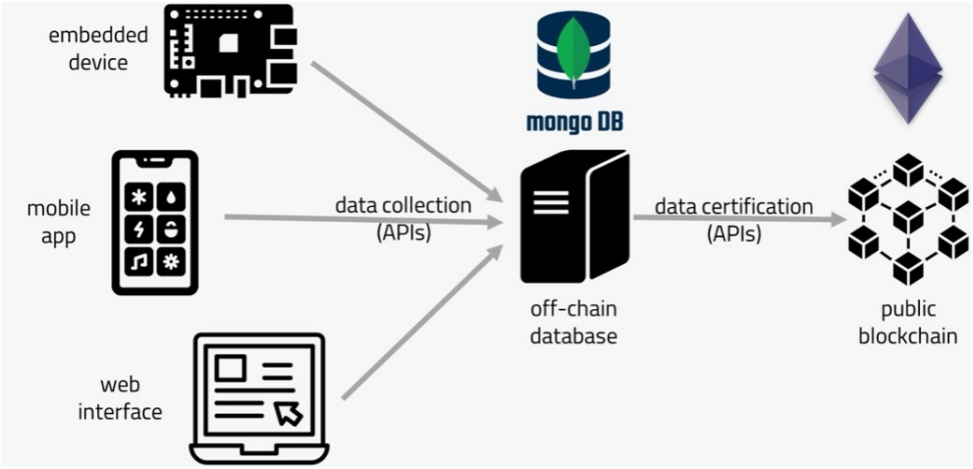

Figura 1: Schema dell'applicazione.

In generale i dati dovranno essere caricati dalle aziende partecipanti che, tramite form dipendenti dal contesto applicativo, potranno accedere alla compilazione e al conseguente inserimento dei dati stessi su una base di dati dedicata. I campi previsti potranno essere diversi da contesto a contesto e potranno riguardare aspetti diversi

dell'attività svolta dall'azienda. Dovrà inoltre essere data la possibilità di caricare file di varia estensione. Il caricamento delle informazioni nel database non è trattato nel lavoro di tesi; infatti, nella simulazione sviluppata nella fase di testing i dati che popolano il database sono scelti in modo pseudo-randomici. Il contributo di questa tesi risiede nella comunicazione tra il database, contenente le informazioni da certificare, e la rete Ethereum, al fine di cristallizzare in modo immutabile il dato.

In primo luogo è stata sviluppata la soluzione che prevede l'inserimento nel campo "data" di una transazione Ethereum dell'hash del dato da certificare per poi passare ad una soluzione che prevede l'organizzazione dei dati in uno o più Merkle tree, in modo da limitare il numero di transazioni da inviare in chain, poiché viene creata una transazione per ogni albero, contenente la propria Merkle root. I dettagli di queste soluzioni saranno esplorati in modo approfondito nel corso di questo lavoro. Intuitivamente, è ragionevole aspettarsi che l'utilizzo dei Merkle tree possa contribuire a limitare i costi in termini di invio di transazioni ma comporterà un aumento di storage richiesto nel database e anche un aumento dei tempi di verifica, in quanto risulterà necessario il calcolo locale della Merkle root. Per questo, oltre ad esporre l'analisi delle prestazioni della simulazione, è stata definita una funzione costo che può essere modellata in modo da adattarsi a molteplici casi di studio teorici in cui i vari contributi che la compongono avranno dei pesi differenti.

L'organizzazione di questa tesi è strutturata per fornire una progressione chiara e logica degli argomenti trattati. Nel Capitolo 1, vengono introdotti i concetti di base partendo dal paradigma DLT per poi concentrarsi sul tema della Blockchain, in particolare sulla rete Ethereum di cui verrà esaminata la struttura delle transazioni. Il Capitolo 2 è dedicato allo studio dello stato dell'arte dei progetti attualmente esistenti relativamente alle tematiche sopramenzionate, andando ad individuare le tecnologie utilizzate, i pregi e i difetti di questi approcci. Il Capitolo 3 comprende la descrizione delle tecnologie chiave utilizzate nel progetto quali il linguaggio di programmazione e le relative librerie, il database e l'ambiente di simulazione della rete Ethereum. Nel Capitolo 4 si procede in primo luogo alla descrizione delle caratteristiche della funzione hash e della sua storia; successivamente, viene presentata la struttura del database con la spiegazione dei campi utilizzati e per concludere viene proposta la prima soluzione per la certificazione dei dati. Nel Capitolo 5 è presente un'introduzione teorica ai Merkle tree, per poi concentrarsi sulla soluzione di certificazione proposta basata su tale struttura dati. Nel Capitolo 6 si offre una dettagliata analisi sui risultati della simulazione condotta per valutare le prestazioni e i costi dei diversi approcci. Successivamente, viene introdotta una funzione costo generale di cui si va ad individuare il punto di minimo, determinando così la soluzione ottimale al variare dei pesi dei vari contributi. Nell'ultimo capitolo, il Capitolo 7, si conclude il presente studio, riassumendo i risultati più importanti di questa tesi.

Capitolo 1

Distributed Ledger Technology

La Distributed Ledger Technology (DLT) [1], in italiano "tecnologia dei registri distribuiti", è un paradigma tecnologico che sottostà ai sistemi di registrazione distribuita, ovvero sistemi in cui tutti i nodi di una rete possiedono la medesima copia di un database che può essere letto e modificato in modo indipendente dai singoli nodi. A differenza dei tradizionali registri centralizzati, la DLT si basa su una rete di computer distribuiti, noti come nodi, che lavorano insieme per registrare, convalidare e conservare i dati in modo decentralizzato.

Uno dei principi fondamentali della DLT è la decentralizzazione: invece di affidarsi ad un'autorità centrale o a un intermediario per gestire le transazioni e mantenere i registri, la DLT consente a ogni nodo della rete di partecipare attivamente al processo. Questo rende la DLT resistente ad un singolo punto di fallimento e fornisce una maggiore sicurezza e affidabilità dei dati.

Un altro aspetto chiave della DLT è l'immutabilità dei dati. Una volta che una transazione o un dato viene registrato nella DLT, diventa permanente e immutabile, non soggetto a modifiche o eliminazioni senza il consenso della maggioranza dei nodi della rete. Eventuali modifiche sostanziali richiedono un consenso generalizzato, il che può portare a una divergenza temporanea o permanente nella catena, nota come fork, dove la comunità deve scegliere quale percorso seguire in base alle nuove regole proposte. Una fork si può verificare anche quando due o più nodi producono blocchi simultaneamente e tentano di aggiungerli alla catena; questi blocchi concorrenti sono collegati allo stesso blocco ma sono diversi nei contenuti. Nella risoluzione della biforcazione la catena che guadagna il consenso diventa la catena principale mentre l'altra catena viene abbandonata. Ciò garantisce che i dati registrati nella DLT siano sicuri, verificabili e resistenti a manipolazioni [2]. Come tale, la tecnologia a registro distribuito elimina la necessità, per le entità che utilizzano il registro, di fare affidamento su un'autorità centrale di fiducia che controlla il registro come invece avviene nei sistemi centralizzati (Figura 1.1).

Un attributo distintivo dei sistemi DLT è la struttura del registro: la soluzione in cui il registro è strutturato come una catena di blocchi contenenti più transazioni prende il nome di Blockchain.

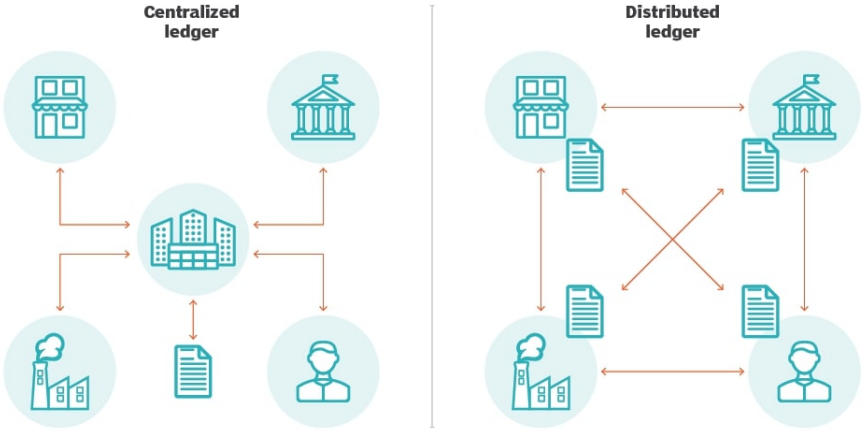

Figura 1.1: Architetture di un sistema centralizzato e distribuito [3].

1.1 Blockchain

La blockchain [4] è una struttura dati che consiste in elenchi crescenti di record, denominati "blocchi", collegati tra loro in modo sicuro utilizzando la crittografia. Ogni blocco contiene un hash crittografico del blocco precedente, un timestamp e dati di transazione. Poiché ogni blocco contiene informazioni sul blocco precedente, questi formano effettivamente una catena con ogni blocco aggiuntivo che si collega a quelli precedenti (Figura 1.2).

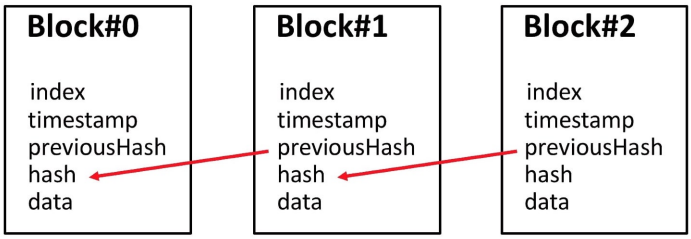

Figura 1.2: Struttura dati di una blockchain [4].

Di conseguenza, le transazioni blockchain sono irreversibili in quanto, una volta registrate, i dati in un determinato blocco non possono essere modificati retroattivamente senza alterare tutti i blocchi successivi.

Esistono diverse tecnologie di blockchain [5]; nel seguito, si riportano alcune dei paradigmi più diffusi ed utilizzati.

- *Blockchain pubblica*: è una rete in cui tutti i partecipanti hanno accesso al registro distribuito e possono partecipare alla verifica e alla validazione delle transazioni. Questo tipo di blockchain è aperto a tutti e offre un alto grado di trasparenza e immutabilità.

- *Blockchain privata*: è un tipo di DLT in cui una singola organizzazione governa la rete, controllando chi è autorizzato a partecipare, eseguire un protocollo di consenso e mantenere il registro condiviso. Queste entità possono essere organizzazioni, aziende o gruppi di partecipanti che hanno interesse a condividere informazioni e transazioni tra di loro in modo sicuro e privato. Le blockchain private offrono maggiore controllo e riservatezza rispetto alle blockchain pubbliche, ma possono perdere un po' di trasparenza, immutabilità e in termini di decentralizzazione.

- *Blockchain autorizzata*: è un ibrido tra la blockchain pubblica e quella privata. In una blockchain autorizzata, l'accesso al registro è limitato solo ad alcune entità autorizzate, ma la verifica e la validazione delle transazioni sono gestite da un gruppo selezionato di partecipanti. Questo tipo di blockchain è spesso utilizzato in settori dove è necessario garantire l'identità e la reputazione dei partecipanti.

1.1.1 Bitcoin

La tecnologia blockchain è nata con l'introduzione di Bitcoin nel 2009 da parte di un individuo o gruppo di persone conosciuto con lo pseudonimo Satoshi Nakamoto. Bitcoin [6] è stata la prima applicazione concreta della tecnologia blockchain, è stata progettata come una valuta digitale peer-to-peer che funziona senza l'intermediazione di una banca centrale o un'autorità finanziaria con l'obiettivo di creare un sistema di trasferimento di valore sicuro, trasparente e decentralizzato. Il logo di bitcoin è mostrato in Figura 1.3.

Figura 1.3: Logo di Bitcoin [6].

Il concetto chiave introdotto da Bitcoin è la creazione di un registro pubblico e distribuito che contenga tutte le transazioni effettuate con la criptovaluta. Questo registro è accessibile a tutti e ogni blocco (perciò, ogni transazione) è prodotto dalla rete tramite un processo chiamato mining. I miner, mediante la risoluzione di

complessi puzzle crittografici, validano e raggruppano le transazioni in blocchi che vengono poi aggiunti alla chain.

1.2 Ethereum

Ethereum [7] è stata concepita da Vitalik Buterin, un programmatore che faceva parte della comunità Bitcoin e ne aveva una profonda conoscenza, inclusa la necessità di generalizzare l'infrastruttura blockchain esistente.

La blockchain di Bitcoin stava iniziando a essere utilizzata per scopi diversi dal semplice trasferimento di denaro, ad esempio per la scrittura di messaggi all'interno della blockchain. In tali casi, si generavano transazioni che non coinvolgevano il trasferimento di denaro o coinvolgevano solo una quantità minima, poiché l'obiettivo principale era scrivere dati nella blockchain anziché effettuare transazioni monetarie. Da questa esigenza è emersa la necessità di sviluppare una blockchain più generale, in grado di supportare nativamente applicazioni diverse dal semplice trasferimento di denaro.

Vitalik Buterin ha colto questa necessità fin dalle prime fasi e nel 2013 ha redatto un white paper che descriveva una nuova piattaforma chiamata Ethereum [8]. L'obiettivo principale di Ethereum era quello di fornire una piattaforma in grado di supportare un modello di blockchain più generale, in grado non solo di scrivere dati arbitrari nella blockchain utilizzando un linguaggio di scripting generale, ma anche di eseguire applicazioni, ovvero programmi decentralizzati. In questo modo, Ethereum andava oltre il concetto di una semplice repository di dati immagazzinati nella blockchain, in quanto i nodi di mining diventavano anche i garanti dell'esecuzione permanente di programmi scritti all'interno della blockchain.

Ethereum è stata perciò sviluppata per affrontare la necessità di una blockchain più generale che potesse supportare applicazioni e programmi decentralizzati. Ciò ha aperto la strada a una vasta gamma di applicazioni decentralizzate e ha trasformato la blockchain in una piattaforma per l'esecuzione di programmi e la gestione di dati immutabili.

1.2.1 Il funzionamento della rete Ethereum

Ogni azione su Ethereum richiede risorse computazionali, come la memorizzazione dei dati di una transazione o l'esecuzione di uno smart contract. Per garantire che queste attività vengano elaborate in modo affidabile e sicuro, è necessario allocare risorse sotto forma di "gas". Inizialmente, Ethereum adottò il meccanismo di consenso chiamato Proof of Work (PoW), noto per essere utilizzato anche da Bitcoin. In PoW, i partecipanti, chiamati "minatori", risolvono complessi problemi matematici. Questo processo richiede una quantità significativa di potenza computazionale e energia elettrica. Il primo minatore che risolve con successo il problema può scrivere il nuovo

blocco sulla blockchain e ricevere una ricompensa in criptovaluta. Più lavoro devono svolgere i minatori, più gas richiedono.

Tuttavia ora Ethereum sta utilizzando un diverso meccanismo di consenso chiamato Proof of Stake (PoS) [9]. In PoS, invece di risolvere puzzle matematici, i partecipanti, chiamati "validatori", vengono selezionati per creare un nuovo blocco in base alla quantità di criptovaluta che possiedono e hanno bloccato come garanzia. In questo sistema, maggiore è la quantità di criptovaluta posseduta e "bloccata", maggiore è la probabilità di essere scelti per validare una transazione e ottenere ricompense. Questo meccanismo è più efficiente in termini energetici rispetto al PoW e promuove la partecipazione responsabile nella rete Ethereum.

Ci sono due distinte unità di misura nel contesto delle transazioni su blockchain. Da un lato, c'è l'Ether, una criptovaluta con il proprio mercato e fluttuazioni di prezzo. L'Ether può essere scambiato con altre criptovalute o con valuta tradizionale. Dall'altro lato, c'è il gas, che rappresenta l'unità di misura per lo sforzo computazionale richiesto per eseguire un'azione sulla blockchain. Il gas non è influenzato dal mercato, ma è strettamente correlato al lavoro computazionale necessario sulla blockchain.

Struttura transazioni L'architettura di una transazione [10] è costituita da una serie di parametri, che vengono scelti a seconda del formato di transazione che si vuole implementare. La struttura contiene i seguenti campi:

- `Nonce`: un contatore con incremento sequenziale, che indica il numero della transazione dall'indirizzo.
- `To`: l'indirizzo del destinatario della transazione.
- `GasPrice`: quantità scalare pari al numero di Wei (1 Ether corrisponde a 10^{-18} Wei) da pagare per unità di gas per tutti i costi di calcolo sostenuti a seguito dell'esecuzione della transazione.
- `GasLimit`: l'importo massimo di unità di gas consumabili dalla transazione.
- `Value`: quantità di ETH da trasferire dal mittente al destinatario.
- `Data`: campo opzionale per includere dati arbitrari.
- `v`: valore indicato come parametro di recupero della chiave pubblica e viene calcolato dopo aver ricavato `r` ed `s`.
- `r`, `s`: valori corrispondenti alla firma della transazione e utilizzati per determinare il mittente della transazione.

Se il costo totale reale supera quello che l'utente è disposto a sostenere per la transazione, quest'ultima può essere rifiutata o considerata non valida. L'utente potrebbe essere tenuto a coprire eventuali costi già sostenuti dai validatori durante

l'esecuzione parziale della transazione. Questo sistema mira a prevenire l'abuso della rete e ad assicurare che le operazioni siano eseguite in modo efficiente; incoraggia gli utenti ad allocare risorse adeguate per garantire che le loro transazioni siano elaborate senza interruzioni.

1.2.2 Smart contract

Gli smart contract [11] sono gli elementi fondamentali delle applicazioni Ethereum. Sono programmi memorizzati sulla blockchain che permettono di convertire i contratti tradizionali in contratti digitali paralleli. Gli smart contract sono estremamente logici, seguendo una struttura "se questo, allora quello". Ciò significa che si comportano esattamente come programmato e non possono essere modificati.

Gli smart contract sono, di fatto, un account su Ethereum, in quanto possiedono un bilancio e la capacità di ricevere transazioni. Tuttavia, a differenza degli account degli utenti, non sono controllati direttamente da un singolo individuo, infatti, essi vengono distribuiti sulla rete Ethereum e eseguiti secondo il codice con cui sono stati programmati. Gli utenti possono interagire con gli smart contract inviando transazioni che eseguono le funzioni specificate nel contratto. Queste transazioni possono coinvolgere l'invio di valuta digitale o l'accesso e la modifica dei dati presenti nello smart contract.

Quando uno smart contract è stato distribuito sulla blockchain di Ethereum, non può essere eliminato o modificato. Le interazioni con gli smart contract sono immutabili e permanenti, il che significa che tutte le transazioni e le modifiche di stato effettuate all'interno dello smart contract sono visibili e non possono essere annullate.

Contratti Multisig I multisig (multiple-signature) contracts rappresentano una categoria di smart contracts che richiede la presenza di più firme valide per eseguire una transazione. I contratti Multisig dividono anche la responsabilità dell'esecuzione del contratto e della gestione delle chiavi tra più parti e prevengono la perdita di una singola chiave privata che potrebbe portare alla perdita irreversibile dei fondi. Per questi motivi, i contratti Multisig possono essere utilizzati per la governance di DAO (Decentralized Autonomous Organization) semplici.

I contratti Multisig richiedono N firme su un totale di M firme possibili (dove $N \leq M$ e $M > 1$) per eseguire una transazione. I valori comunemente usati sono quelli in cui N corrisponda alla maggioranza, ovvero, $N = \lceil M/2 \rceil$; valori tipici per M sono 5 e 7. In altre parole, valori tipicamente utilizzati sono $N = 3$ ed $M = 5$, e $N = 4$ ed $M = 7$. Ad esempio, un contratto Multisig 4/7 richiede quattro firme valide su sette possibili: ciò significa che i fondi possono ancora essere recuperati anche se tre firme vengono perse. In questo caso, ciò comporta che la maggioranza dei titolari di chiavi deve essere d'accordo e firmare affinché il contratto venga eseguito.

1.2.3 I Token Ethereum

Una delle funzioni implementate da Ethereum tramite gli smart contract è la creazione di nuove criptovalute, che hanno avuto un grande successo all'interno dell'ecosistema Ethereum. Grazie agli smart contract, è possibile generare nuove criptovalute chiamate token.

I token possono essere creati in qualsiasi quantità e si prestano perciò ad essere utilizzati come mezzi di scambio per beni e servizi generici. Inizialmente, venivano scritti smart contract personalizzati per creare queste nuove criptovalute, ma nel tempo sono emersi degli smart contract modello che possono essere adattati per qualsiasi criptovaluta, seguendo uno standard specifico.

Un esempio di tali token sono i token ERC721. La peculiarità di questi token ERC721 è che sono non fungibili, da qui il nome NFT (non-fungible token) [12], ossia qualcosa di unico che non può essere sostituito da altro. Ogni token ERC721 è associato a un esemplare unico. L'idea di avere una garanzia crittografica dell'unicità di un esemplare ha aperto nuovi mercati, come quello delle opere d'arte. Le opere d'arte necessitano di una certificazione di unicità, e i token non fungibili forniscono una certificazione crittografica dell'unicità che non può essere alterata. Perciò, un token ERC-721 è un token collezionabile e il suo valore è definito in base alla rarità e particolarità delle sue proprietà.

In questo contesto, l'utilizzo dei token non fungibili rappresenta uno strumento innovativo che consente di assegnare agli utenti oggetti crittografici unici, la cui garanzia non è affidata a un proprietario, a un fornitore di servizi o a una multinazionale, ma alla blockchain stessa, che è di dominio pubblico. Gli NFT trovano impiego in progetti che gestiscono risorse uniche sulla blockchain, e le possibili applicazioni sono estremamente diverse, spaziando dal diritto d'autore alla vendita dei biglietti, dai certificati all'autenticazione fino all'arte digitale.

Capitolo 2

Studio dello stato dell'arte

In questo capitolo verrà presentata una panoramica sui principali progetti e pubblicazioni inerenti ai temi della tracciabilità, principalmente alimentare, e della certificazione dei dati che facciano uso della tecnologia Blockchain.

2.1 Tracciabilità

Le catene di approvvigionamento sono estremamente complesse e si prevede che tale complessità crescerà sempre più negli anni. Questa complessità deriva dal fatto che le materie prime attraversano in molti casi lunghi processi di trasformazione e viaggiano attraverso diversi Paesi prima di essere trasformate in prodotti finiti per i consumatori. Pertanto, diventa essenziale disporre di un sistema di tracciabilità che segua l'intero ciclo di vita di un prodotto, dalla sua origine fino alle tavole dei consumatori.

Il processo di tracciabilità si articola in diverse fasi, che includono l'identificazione, l'acquisizione, la registrazione, la gestione e l'elaborazione dei dati, nonché la trasmissione e la comunicazione degli stessi. La blockchain emerge come una tecnologia intrigante per la condivisione di informazioni nel settore alimentare, garantisce un ambiente affidabile ma va combinata con altre soluzioni tecnologiche atte a rafforzare il legame tra mondo fisico e mondo digitale. Tra questi accorgimenti si annoverano soluzioni IoT (Internet of Things), quali sensori RFID (Radio Frequency Identification)e NFC (Near Field Communication).

2.1.1 Food Trust

In [13] viene presentata una panoramica sui progetti legati al mondo del food tracking partendo dal sistema "Food Trust", uno dei principali sistemi di tracciabilità alimentare basati su blockchain sviluppato da IBM nel 2017 [14]. Food Trust non solo contribuisce alla sicurezza alimentare, ma offre anche benefici ai produttori, come la freschezza degli alimenti, la sostenibilità e la riduzione dello spreco. IBM ha annunciato che oltre 5 milioni di prodotti alimentari già presenti sul mercato sono inclusi nel sistema.

La soluzione offre agli utenti autorizzati un accesso immediato ai dati della catena di approvvigionamento alimentare, dall'azienda agricola al negozio e, in ultima analisi,

al consumatore. I prodotti alimentari vengono identificati con codici univoci, come il Numero Globale di Articolo Commerciale (GTIN) o il Codice a Barre Universale (UPC). L'architettura non si basa su Ethereum ma su Hyperledger Fabric.

La piattaforma Hyperledger Fabric [15] è un framework blockchain open-source permissioned fondato dalla Linux Foundation, il che significa che, a differenza di una rete pubblica permissionless, i partecipanti sono noti tra loro anziché essere anonimi e quindi non attendibili. Uno dei più importanti fattori di differenziazione della piattaforma è il suo supporto per protocolli di consenso intercambiabili che consentono alla piattaforma di essere personalizzata più efficacemente per adattarsi a casi d'uso particolari. Le informazioni chiave relative al prodotto, come la provenienza, le informazioni sulla qualità, le certificazioni e le date di produzione, vengono registrate sulla blockchain di IBM Food Trust. Queste informazioni sono immutabili e accessibili a tutti i partecipanti autorizzati della catena di approvvigionamento.

La piattaforma di IBM Food Trust viene eseguita su infrastrutture cloud fornite da IBM: l'utilizzo di un'architettura cloud consente l'accesso e la condivisione dei dati da parte di tutti i partecipanti autorizzati in modo rapido e conveniente, eliminando la necessità di installare e gestire sistemi locali complessi. Inoltre, l'architettura cloud offre scalabilità, consentendo al sistema di gestire un grande volume di dati e di adattarsi alle esigenze in continua evoluzione delle aziende alimentari. Ad esempio, Walmart ha utilizzato il servizio proposto da IBM per registrare l'origine di ogni pezzo di carne acquistato dalla Cina, il processo di lavorazione, lo stoccaggio e tutte le transazioni correlate alla vendita, insieme al percorso storico del prodotto [16].

2.1.2 Tracciabilità agricola per HACCP

In [17] viene proposta una soluzione blockchain per la tracciabilità agricola con lo scopo di garantire che i principi e i requisiti HACCP (Hazard Analysis and Critical Control Points) siano rispettati durante la produzione, il trasporto e la conservazione di un prodotto.

Come mostrato in Figura 2.1, il sistema proposto è un tipico sistema distribuito decentralizzato, che utilizza Internet of Things (come RFID, WSN, GPS) per raccogliere e trasferire e si basa su BigchainDB per memorizzare e gestire i dati dei prodotti nella supply chain alimentare. BigchainDB è un database decentralizzato che combina le tradizionali funzionalità di database e le funzionalità blockchain (decentralizzazione, immutabilità). Come database, BigchainDB è complementare ad altri sistemi decentralizzati, come l'archiviazione decentralizzata dei file (IPFS), protocolli per lo scambio di dati e blockchain per smart contract (Ethereum o Hyperledger). La Figura 2.2 illustra alcuni modalità in cui BigchainDB potrebbe essere utilizzato [18].

Nella catena di fornitura, ci sono diversi partecipanti come fornitori, produttori, distributori, rivenditori, consumatori e certificatori. Ognuno di questi partecipanti può gestire le informazioni sui prodotti in BigchainDB se sono registrati come utenti

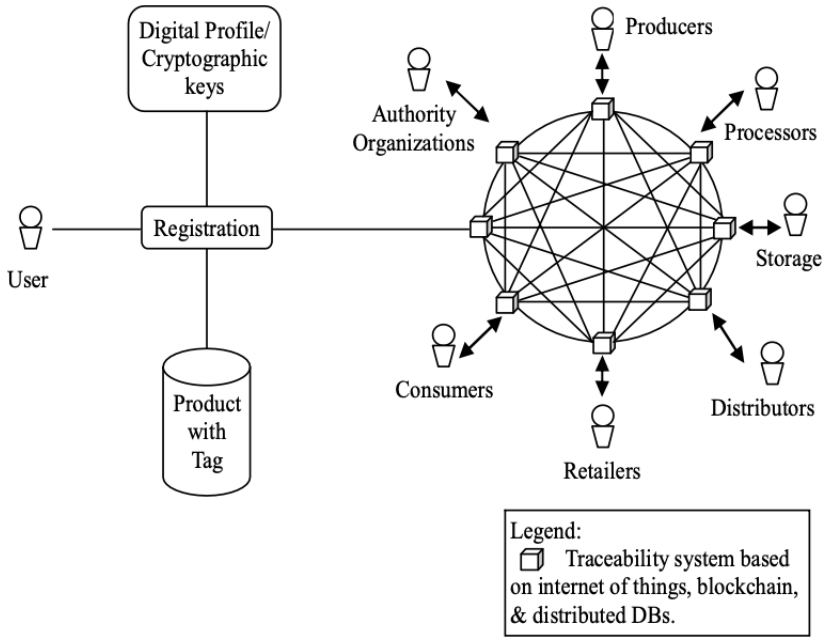

Figura 2.1: Framework del sistema di tracciabilità [17].

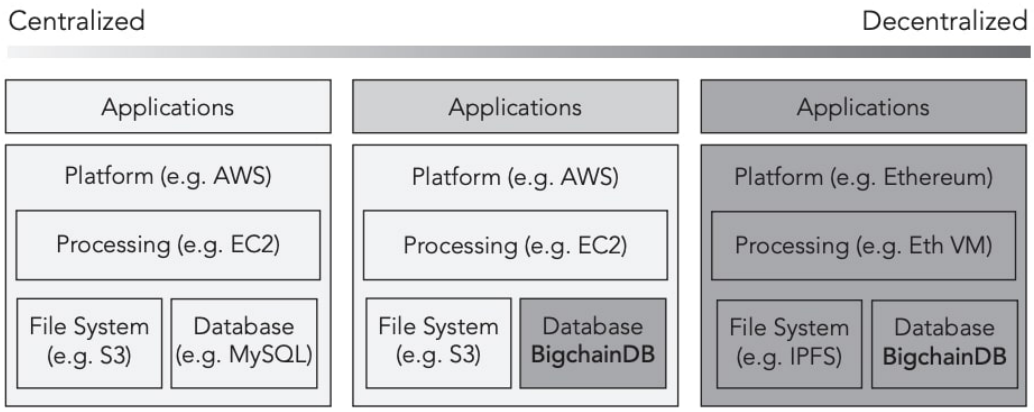

Figura 2.2: Possibili utilizzi: da schemi centralizzati a decentralizzati [17].

nel sistema. A ciascun prodotto viene assegnato un tag (RFID), che è un identificatore crittografico digitale univoco che collega gli oggetti fisici alla loro identità virtuale nel sistema. Questa identità virtuale rappresenta il profilo informativo del prodotto. I partecipanti della catena di fornitura possono registrarsi come utenti nel sistema per ottenere un'identità unica. Dopo la registrazione, viene generata una coppia di chiavi crittografiche pubbliche e private per ciascun utente. La chiave pubblica viene utilizzata per identificare l'utente all'interno del sistema, mentre la chiave privata serve per autenticare l'utente durante le interazioni con il sistema.

Nel settore dell'approvvigionamento alimentare, quando un utente in un determinato collegamento riceve un prodotto, solo quell'utente può aggiungere nuovi dati al

profilo del prodotto utilizzando la propria chiave privata. Inoltre, quando l'utente trasferisce il prodotto al successivo, entrambi devono firmare un contratto digitale per autenticare lo scambio. I dettagli della transazione vengono quindi registrati in BigchainDB e il sistema elabora e aggiorna automaticamente i dati nel profilo del prodotto, consentendo agli utenti di condividere lo stato dei prodotti in qualsiasi momento. Alcuni utenti possono desiderare di mantenere alcune informazioni private e segrete. Dal punto di vista tecnologico, il sistema può proteggere le identità mentre trasferisce altre informazioni rilevanti. Ad esempio, i produttori nella catena di fornitura possono stipulare contratti digitali con gli utenti successivi, mantenendo al contempo la loro identità privata. Nel sistema sono presenti autorità di terze parti, ma la differenza è che anche loro hanno profili digitali: controllano e verificano l'identità e il comportamento degli utenti, registrando i risultati in BigchainDB.

2.1.3 Agri-BlockIoT

Agri-BlockIoT è un ulteriore progetto che propone una soluzione nel campo agricolo basata su blockchain [19]. Viene implementata un'architettura a strati (Figura 2.3) in grado di fare affidamento sulle tecnologie Blockchain e IoT per ottenere la trasparenza, la verificabilità e l'immutabilità dei registri memorizzati.

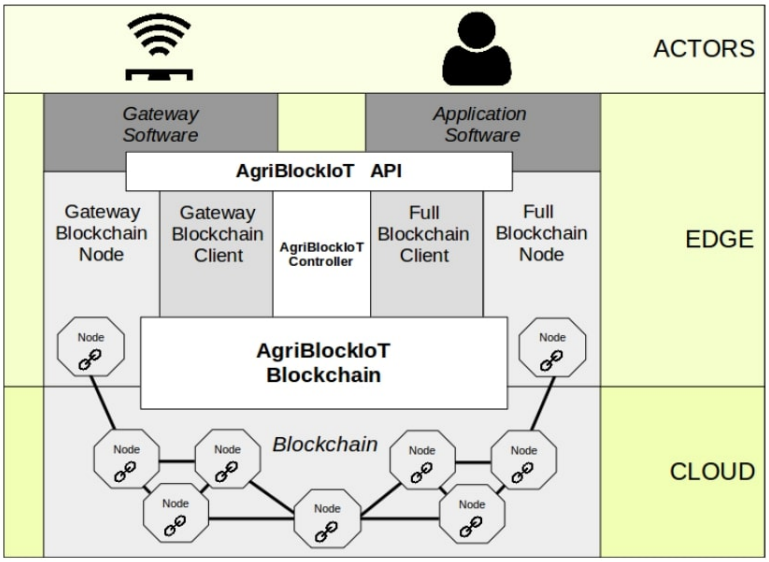

Figura 2.3: Architettura Agri-BlockIoT [19]

L'architettura proposta sfrutta i moderni dispositivi di edge computing, come gateway e mini-PC, come nodi della blockchain. Ciò aumenta la resistenza, la decentralizzazione, la sicurezza e la fiducia nella rete.

I principali moduli di AgriBlockIoT sono:

- `API`: un'interfaccia di programmazione delle applicazioni REST che espone le funzionalità di AgriBlockIoT ad altre applicazioni. Consente un'agevole

integrazione con i sistemi software esistenti.

- `Controller`: un componente che trasforma le chiamate di funzioni di alto livello nelle corrispondenti chiamate di basso livello per il layer blockchain e viceversa. Gestisce l'interrogazione e la conversione dei record di dati memorizzati nella blockchain in informazioni di alto livello per il livello superiore.

- `Blockchain`: il componente principale del sistema che contiene tutta la logica di business implementata tramite smart contract sulla blockchain. La complessità di questo modulo varia a seconda della blockchain selezionata e delle capacità del programma e delle interfacce client.

È necessario che tutti i partecipanti, compresi i dispositivi IoT, siano utenti registrati della blockchain sottostante e dispongano delle corrette coppie di chiavi pubbliche/private per firmare digitalmente le operazioni nel registro distribuito. Per valutare le prestazioni di AgriBlockIoT, è stato implementato il funzionamento di un dispositivo IoT che produce valori digitali archiviati direttamente nella blockchain.

I dati archiviati possono essere recuperati, e possono essere implementati smart contract che vengono eseguiti in modo autonomo in base a determinate condizioni sui dati del sensore. Sono state utilizzate due implementazioni di reti private, Ethereum e Hyperledger Sawtooth, per testare AgriBlockIoT. La scelta di queste implementazioni è dovuta alla diversa personalizzazione dei record nel registro (transazioni). Ethereum utilizza una struttura di transazione unica, mentre Hyperledger Sawtooth consente la definizione di una struttura di transazione personalizzata.

2.1.4 Blockchain for food tracking

Infine, in [13] viene proposta un'architettura di sistema basata su blockchain per il tracciamento alimentare, come mostrato in Figura 2.4.

A differenza di un tradizionale schema di tracciamento (Figura 2.5) l'introduzione del blockchain layer permette la creazione di un sistema non controllato da un'autorità centrale, perché la proprietà dei dati non appartiene a nessuna parte, in quanto tutti i partecipanti hanno una copia dei dati in una struttura distribuita. La distribuzione dei dati ottenuti sarà consegnata all'utente finale attraverso i server (Figura 2.6). Dopo che i dati ricevuti dalle unità della supply chain sono passati attraverso il livello blockchain, vengono creati smart contract e tutte le transazioni vengono eseguite e monitorate sempre attraverso smart contract. I dati ottenuti dall'output del processo vengono inviati a livello Internet per la distribuzione.

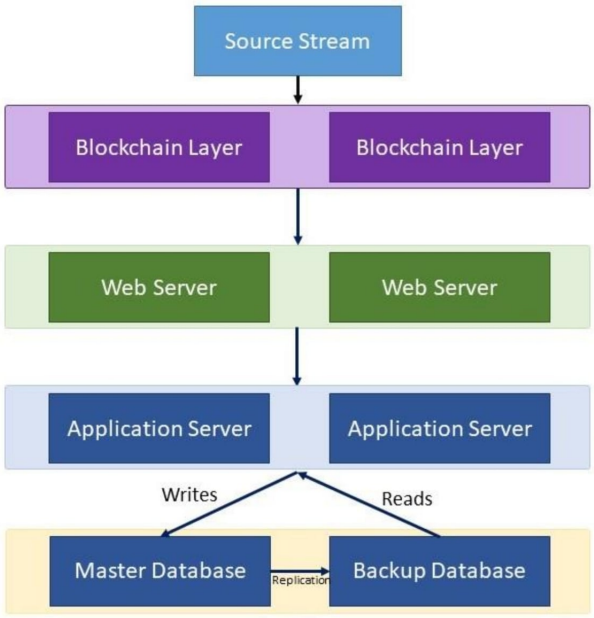

Figura 2.4: Architettura basata su blockchain per il tracciamento alimentare [13].

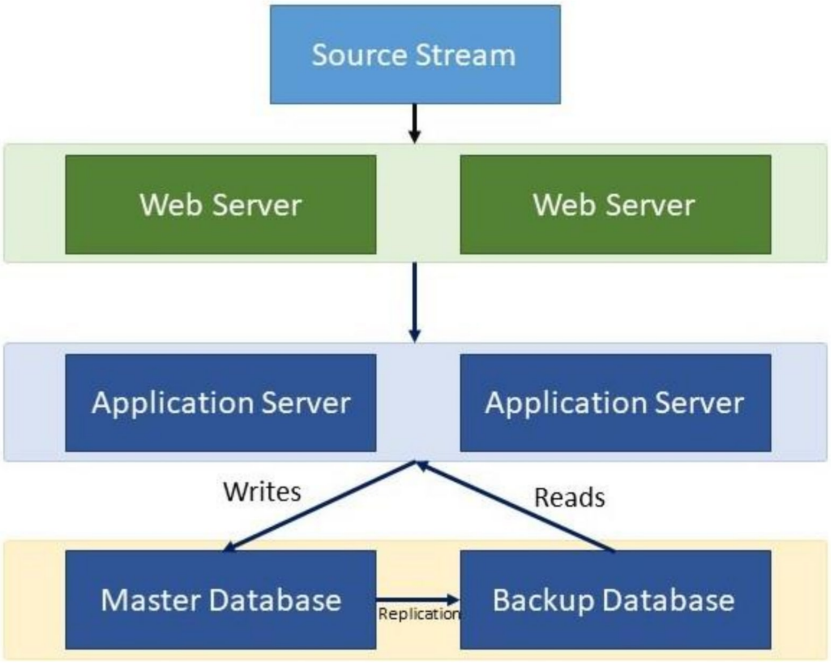

Figura 2.5: Architettura di sistemi di tracciamento alimentare convenzionale [13].

È stato deciso di utilizzare l'infrastruttura blockchain Hyperledger Fabric all'interno di questo studio, e non Ethereum, per un motivo di performance in termini di latenza, trasmissione/ricezione al secondo e tassi di carico della CPU, come si evince dalla Figura 2.7.

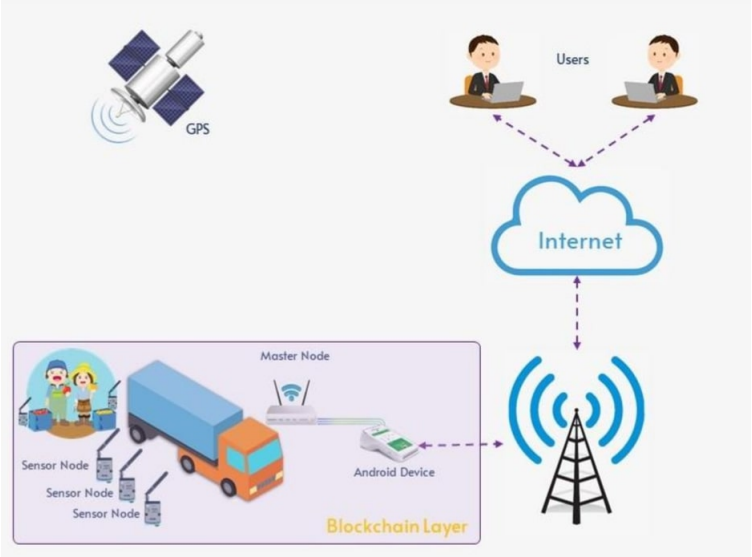

Figura 2.6: Sistema proposto [13].

	Latency (s)	Net Tx (Bytes)	Net Rx (Bytes)	CPU Load (%)
Ethereum	16.55	528	682	46.78
Hyperledger	0.021	19	20	6.75

Figura 2.7: Confronto Ethereum e Hyperledger [13].

2.2 Certificazione

In questa sezione verranno presentati i progetti sviluppati finora relativamente alla certificazione dei dati in blockchain, ovvero come sfruttare la sue caratteristiche per avere dei dati affidabili, sicuri ed integri. La maggior parte degli esempi di utilizzo si rifà a veri e propri certificati accademici; perciò, tali esempi ci spostano sempre di più verso il concetto di proof of attendance.

2.2.1 BlockIPFS

Il primo documento analizzato [20] è focalizzato sulla privacy e sulla sicurezza dei file condivisi; esso ha l'obiettivo di integrare l'IPFS con la tecnologia blockchain, proponendo un nuovo approccio, denominato BlockIPFS, che va ad intervenire sulle mancanze del servizio IPFS. BlockIPFS permette di ottenere una maggiore affidabilità dei dati e la protezione dell'autore fornendo un percorso chiaro per rintracciare tutte le attività associate a un dato file.

IPFS è un sistema di archiviazione decentralizzato in cui i contenuti vengono indirizzati utilizzando hash crittografici, garantendo così l'unicità e l'integrità dei dati. Questa tecnologia è particolarmente utile per la distribuzione di contenuti su Internet in modo efficiente e affidabile, evitando la dipendenza da server centralizzati.

IPFS è comunemente utilizzato come piattaforma di archiviazione per la condivisione di dati, ha i vantaggi di alta disponibilità e buone prestazioni, ma manca della capacità di tracciare l'accesso e l'autenticazione, il che rende difficile investigare sugli accessi non autorizzati e sull'autenticità. Nell'IPFS attuale, non esiste un modo per un'organizzazione di creare regole di accesso ai file nella propria rete IPFS privata. Se un file destinato a essere condiviso solo con la direzione viene condiviso sulla rete, qualsiasi membro della rete IPFS dell'organizzazione con accesso all'hash del file può semplicemente accedervi. Perciò, l'IPFS non offre alcuna capacità di tracciabilità per registrare e verificare l'accesso ai file sulla rete.

Il focus principale del lavoro è quello di affrontare il problema della tracciabilità in modo che tutte le attività relative a un oggetto specifico su un sistema di file distribuito possano essere tracciate e controllate in termini di accesso. Nell'articolo viene proposto un nuovo approccio che cerca di migliorare IPFS utilizzando Hyperledger Fabric per fornire una traccia che può essere utilizzata per l'audit e la tutela dell'autenticità dei file. Le operazioni sui file, come l'aggiunta o l'accesso a un file, generano metadati che vengono registrati su una blockchain di Hyperledger Fabric. La blockchain è responsabile per l'archiviazione e la gestione dei metadati del file.

In questa implementazione, quando un utente accede a un file su altri nodi, nessun record viene inviato alla propria blockchain locale. Tuttavia, i log di accesso vengono registrati nel BlockIPFS del proprietario del file. L'utente può recuperare i metadati dal proprio BlockIPFS locale per tracciare le attività relative a un file specifico o a tutti i file che ha aggiunto a IPFS. È importante notare che la blockchain di Hyperledger Fabric in BlockIPFS memorizza solo i metadati dei file a fini di tracciabilità, mentre i file stessi sono ancora gestiti da IPFS. In altre parole, la blockchain è quasi trasparente agli utenti quando leggono/scrivono file, e ogni meccanismo di controllo dell'accesso ai file in IPFS viene ereditato da BlockIPFS. Gli utenti interrogano solo la blockchain per esaminare i metadati di un file, e tali metadati vengono registrati sul registro blockchain in modo crittografato, in modo che solo il proprietario del file o gli utenti autorizzati possano leggerli. I file stessi sono crittografati e gestiti da IPFS, mentre le chiavi per decifrare i file sono archiviate nei canali specifici sulla blockchain. Lo schema generale è mostrato in Figura 2.8.

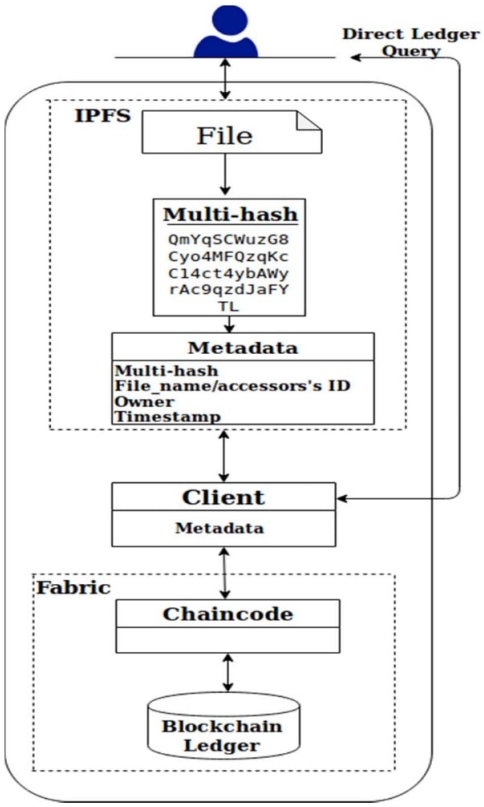

Figura 2.8: Architettura BlockIPFS [20].

2.2.2 Blockcert

Per quanto riguarda la certificazione, esistono vari software progettati per aiutare le applicazioni a emettere, pubblicare e verificare certificati tra cui spicca Blockcert [21] sviluppato da MIT Learning Lab e Media Lab. Blockcert seleziona una rete blockchain come Ethereum o Bitcoin e utilizza i dati delle transazioni a proprio vantaggio per memorizzare i record di certificazione nel formato JSON DL. Essendo uno standard aperto, consente all'utente di avere il pieno controllo sui propri record. Questa tecnologia crea una nuova infrastruttura di fiducia che sostituisce la necessità di richiedere i record a un'autorità centrale. I record digitali sono registrati in un blocco crittograficamente firmato, a prova di manomissione e condivisibile; questo significa che i record rimangono autenticati, non modificabili e pubblici.

Blockerts funziona creando un file digitale che contiene alcune informazioni di base (nome del destinatario, nome dell'emittente, data di emissione) e mandando in chain la firma del contenuto del certificato. Il sistema tiene traccia di chi ha rilasciato e ricevuto il certificato e convalida il contenuto del certificato. La Figura 2.9 mostra la schema di base dell'architettura.

Una soluzione simile è TrueRec, sviluppata da SAP (Systeme, Anwendungen, Produkte in der Datenverarbeitung) [22].

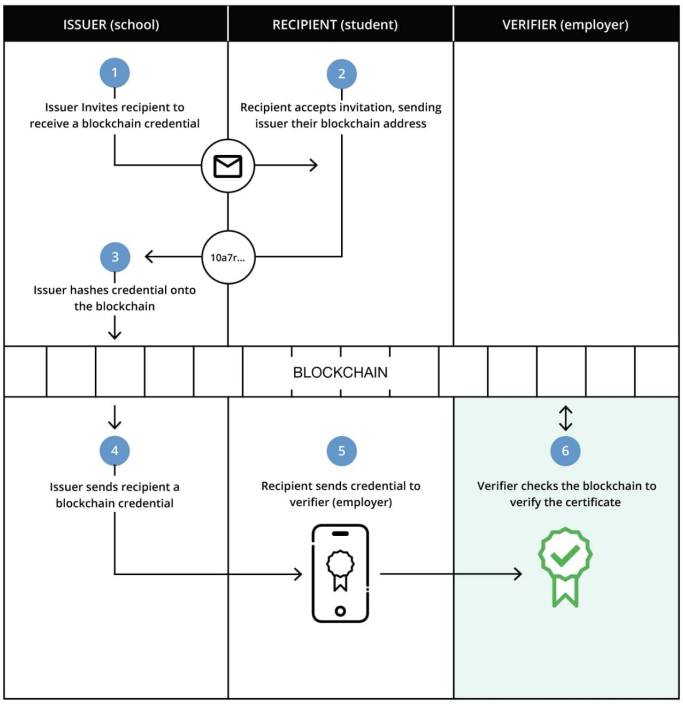

Figura 2.9: Architettura Blockcerts [21].

2.2.3 CertChain

CertChain [23] è una piattaforma di gestione dei certificati che sfrutta la tecnologia blockchain per fornire autenticazione dei certificati e prevenire la contraffazione dei certificati. Utilizza una rete blockchain pubblica con bookkeeper (registri contabili) e autorità di certificazione in ogni nodo. I bookkeeper sono responsabili dell'accesso alla blockchain per registrare le operazioni dei certificati. Il sistema si basa su una architettura a quattro livelli, che comprende il livello dei dati, il livello di rete, il livello di estensione e il livello delle applicazioni (Figura 2.10).

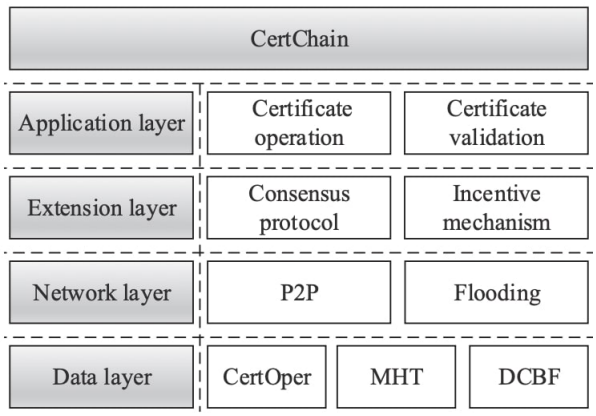

Figura 2.10: Control Service UML [23].

2.2.4 Smart contracts per certificati scolastici

In [24] viene sviluppato un prototipo per la conservazione di certificati sulla blockchain in un formato di immagine digitale. Il servizio progettato dispone di un accesso di autenticazione per garantire che gli inserimenti vengano effettuati solo dalle istituzioni educative autorizzate e che non vengano create inserzioni di certificati da istituzioni false all'interno dell'applicazione. Inoltre, le istituzioni possono modificare le informazioni, a condizione che siano successivamente convalidate sulla blockchain.

Il lavoro si concentra sull'implementazione di una soluzione che utilizza gli smart contracts per emettere e archiviare documenti nella blockchain, in modo da poterli successivamente recuperare quando richiesto. Viene configurata una rete blockchain privata e per gestire tutti i dati tra gli utenti e la blockchain viene implementato un servizio di controllo, che funge da servizio middleware per eseguire le comunicazioni tra input e output. Il servizio di controllo viene visualizzato nel browser, utilizzando pagine web specifiche per garantire determinate azioni. La Figura 2.11 rappresenta il diagramma UML delle funzionalità del servizio di controllo del prototipo.

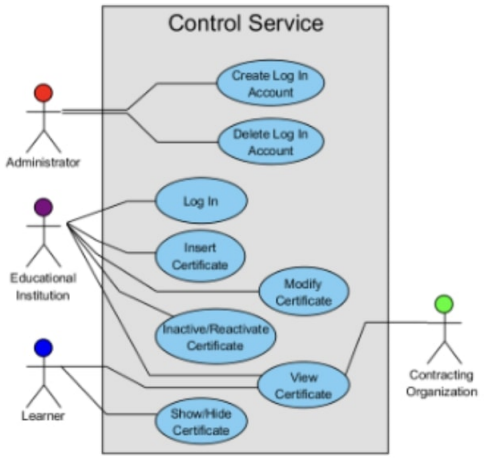

Figura 2.11: Architettura certchain [23].

A causa dei costi elevati della blockchain per lo storage dei file, viene implementato il servizio di archiviazione distribuita IPFS in congiunzione con la blockchain Ethereum, consentendo di archiviare i file (principalmente immagini) e riducendo drasticamente i costi per l'emissione dei contratti. Un database è stato implementato per memorizzare gli hash delle transazioni sulla blockchain al fine di affrontare le limitazioni della ricerca degli account nelle operazioni standard della blockchain. Questo permette una ricerca più rapida e una panoramica dei certificati all'interno di un account specifico.

Inoltre, il database viene utilizzato per archiviare altre informazioni cruciali, come i login degli account delle istituzioni verificate che possono operare nell'applicazione, nonché altre informazioni pertinenti per la gestione del servizio di controllo. Il database è stato creato utulizzando MySQL, un sistema di gestione di database

relazionale open source. Il modello dei dati nella Figura 2.12 mostra le tabelle, i campi e le relazioni del database.

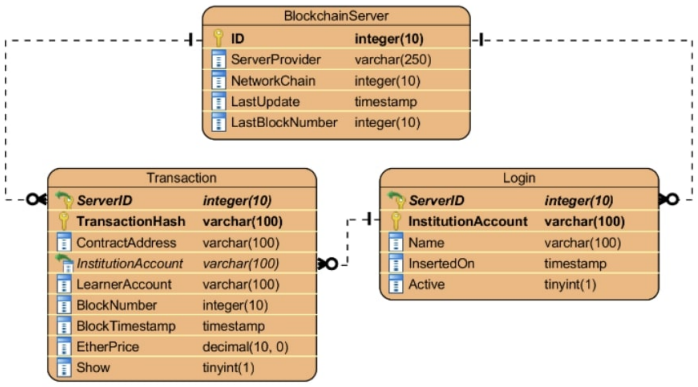

Figura 2.12: Modello dei dati [23].

2.2.5 Schema basato su multi-firma

Basandosi su Blockcerts, l'università di Birmingham [25] ha proposto una serie di soluzioni crittografiche tra cui l'utilizzo di uno schema multi-firma per migliorare l'autenticazione dei certificati e l'applicazione di un meccanismo sicuro di revoca per migliorare l'affidabilità della revoca dei certificati.
Il sistema consiste in quattro componenti: un'applicazione di verifica che include un'identità federata, un'applicazione di emissione che coinvolge la firma multipla e la revoca basata su indirizzo bitcoin, la blockchain e il database locale adottato da MongoDB (Figura 2.13).

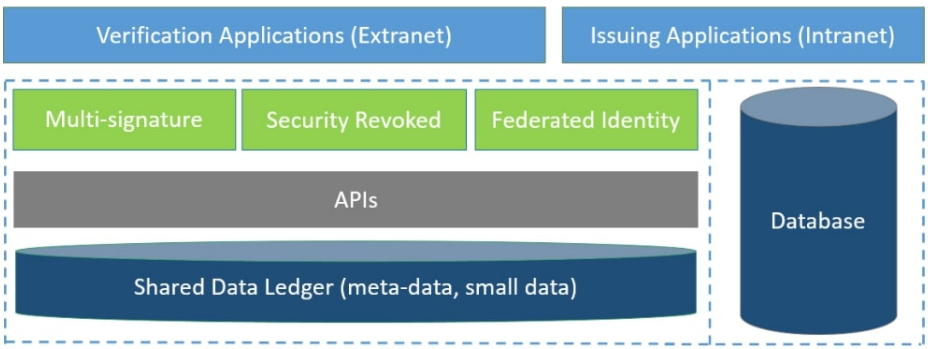

Figura 2.13: Architettura di sistema [25].

Le applicazioni di emissione sono progettate per unire l'hash del certificato in un Merkle tree e inviare la Merkle root alla blockchain insieme alla firma della maggioranza dei membri della comunità. Inoltre, le applicazioni di emissione si occupano della revoca dei certificati. Le applicazioni di verifica si concentrano sulla verifica dell'autenticità e dell'integrità dei certificati emessi. Il meccanismo può essere descritto brevemente nel seguente modo: verifica che il codice di autenticazione sia valido; verifica l'hash con il certificato locale; conferma che l'hash sia nel Merkle tree; assicura che la radice di Merkle sia nella blockchain; verifica che il certificato non sia stato revocato; valida la data di scadenza del certificato.

La blockchain funge da infrastruttura di fiducia e database distribuito per salvare i dati di autenticazione. Tipicamente, i dati di autenticazione consistono nella radice di Merkle generata utilizzando dati hash da migliaia di certificati. MongoDB viene utilizzato come database poiché gestisce con successo i certificati basati su JSON e fornisce un'alta disponibilità e scalabilità. Il database è stato progettato per contenere due categorie di dati: i dati di autenticazione pubblica e i dati dei certificati privati. I dati di autenticazione pubblica sono disponibili per il pubblico e rilasciati alla blockchain; i dati del certificato privato sono memorizzati nel MongoDB dove sono protetti in modo sicuro e isolati nella intranet. Nell'architettura in Figura 2.14, il "server mongo" serve come router per accedere al servizio primario, il "server di configurazione" mantiene il sistema di lavoro metadati e il "mongo server" salva i dati principali.

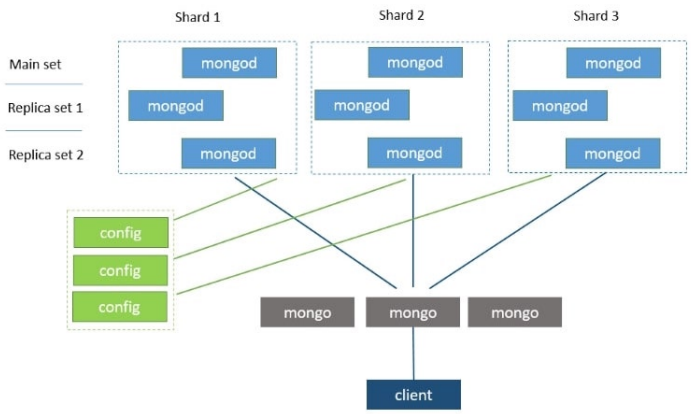

Figura 2.14: Architettura di MongoDB [25].

Lo scenario in Figura 2.15 è semplice: lo studente fa richiesta all'istituto per un certificato e i certificatori verificano le informazioni dello studente e, una volta approvato, lo inseriscono in una transazione BTC. Successivamente, la maggioranza dei membri del comitato accademico lo firma con le proprie chiavi private. Dopo di

ciò, il sistema diffonde la transazione che contiene la Merkle root per tutti i certificati. Una volta che la transazione è confermata, lo studente riceve un certificato in formato JSON.

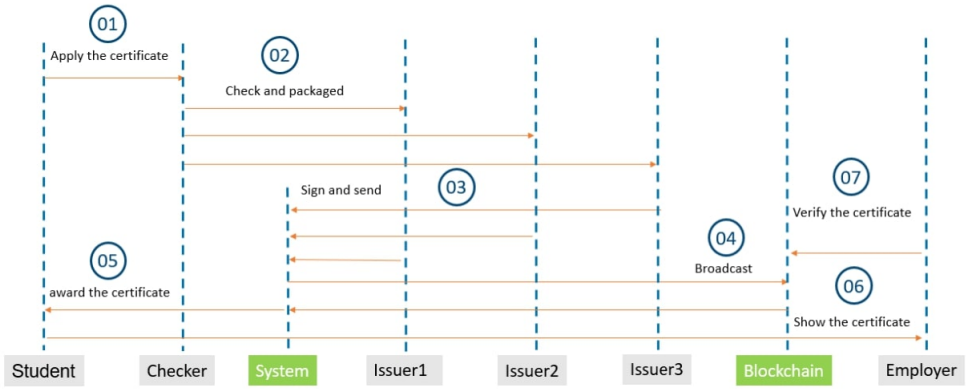

Figura 2.15: Workflow [25] .

Capitolo 3

Tecnologie e strumenti utilizzati

3.1 JavaScript

Il linguaggio JavaScript [26], spesso abbreviato in JS, rappresenta un pilastro fondamentale nel panorama tecnologico del World Wide Web insime a HTML e CSS. Trova applicazione nella creazione di rest API, applicazioni desktop e embedded, siti web e applicazioni web, permettendo la realizzazione di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati dall'utente, come interazioni con mouse, tastiera o caricamento della pagina.

Originariamente concepito da Brendan Eich della Netscape Communications con il nome di Mocha e successivamente ribattezzato "JavaScript", il linguaggio è stato formalizzato con una sintassi più vicina a quella di Java di Sun Microsystems (acquisita da Oracle nel 2010). La sua standardizzazione è avvenuta per la prima volta nel 1997 dalla ECMA con il nome ECMAScript. L'attuale standard, da giugno 2023, è ECMA-262 Edition 14, ed è anche riconosciuto come standard ISO (ISO/IEC 16262). Questa evoluzione ha reso JavaScript un linguaggio altamente flessibile e potente, indispensabile per la creazione di esperienze utente avanzate all'interno di un mondo digitale in costante mutamento.

Nel 2023, il 98,7 % dei siti web fa uso di JavaScript lato client per gestire il comportamento delle pagine web, spesso integrando librerie di terze parti. Originariamente, i motori JavaScript venivano utilizzati solo nei browser web, ma oggi sono componenti centrali di alcuni server e di una varietà di applicazioni. Il sistema di runtime più popolare per questo utilizzo è Node.js. In Figura 3.1 viene mostrato il logo di JavaScript.

Figura 3.1: logo JavaScript [26].

3.1.1 Node.js

Node.js è uno dei framework JavaScript più adottati per lo sviluppo di applicazioni web, questo progetto open source è stato rilasciato nel 2009 e si è rapidamente affermato come uno dei principali runtime JavaScript.Node.js è noto per essere guidato da eventi asincroni ed è costruito sul motore JavaScript V8 sviluppato da Google per il suo browser Chrome. La sua versatilità lo rende utilizzabile sia per la programmazione lato frontend che lato backend.

Una caratteristica distintiva di Node.js è la sua capacità di gestire operazioni di input/output (I/O) in modo non bloccante. Ciò significa che quando vengono eseguite richieste di I/O, come lettura da un file o una richiesta di rete, Node.js non attende passivamente la loro conclusione ma queste operazioni vengono eseguite in background, consentendo al programma principale di continuare ad eseguire altre attività senza interruzioni. Questo approccio asincrono è fondamentale per garantire prestazioni elevate in applicazioni web, dove molte operazioni possono essere gestite simultaneamente.

Node.js fornisce un'ampia gamma di moduli e librerie che aiutano gli sviluppatori a scrivere codice più efficiente, modulare e riutilizzabile e accelerando il processo di sviluppo. Nel progetto sono state utilizzate:

1. `Web3` [27]: è un insieme di librerie le quali permettono di interagire con il proprio nodo blockchain remoto o locale in maniera semplice ed intuitiva tramite l'utilizzo di diversi protocolli come HTTP, IPC o WebSocket. Fornisce funzionalità come la comunicazione con un nodo Ethereum, l'invio di transazioni, la gestione dei contratti intelligenti e l'accesso alle informazioni sulla blockchain.

2. `Crypto` [28]: offre funzionalità di crittografia, come la generazione di hash e l'encrypting/decrypting di dati.

3. `Mongoose` [29]: è uno strumento di modellazione degli oggetti MongoDB per Node.js, consentendo l'interazione con un database MongoDB.

4. `Os-utils` [30]: è una libreria utilizzata per monitorare le risorse di sistema come la CPU e la memoria.

3.2 MongoDB

MongoDB [31] è un popolare sistema di gestione di database orientato ai documenti, progettato per la scalabilità, la flessibilità e le alte prestazioni. Si inserisce nel panorama dei database NoSQL, distinguendosi per la sua capacità di archiviare, interrogare e gestire grandi quantità di dati in modo efficiente e affidabile. A differenza dei tradizionali database relazionali, MongoDB utilizza una struttura di dati basata su documenti JSON flessibili, consentendo una rappresentazione dei dati più dinamica e adattabile alle esigenze delle applicazioni moderne. Questa caratteristica, insieme alla

scalabilità orizzontale e alla facilità di distribuzione, lo rende una scelta popolare per applicazioni web, mobile, IoT e molte altre, offrendo agli sviluppatori un'esperienza intuitiva e potente nella gestione dei dati.

3.2.1 MongoDB Compass

MongoDB Compass è un'applicazione di interfaccia grafica (GUI) intuitiva e versatile progettata per semplificare l'interazione e la gestione di database MongoDB. Rappresenta un'interfaccia utente visivamente accattivante e funzionale, consentendo agli sviluppatori e agli amministratori di database di esplorare, interrogare e manipolare i dati del database in modo efficace. Con MongoDB Compass, gli utenti possono navigare attraverso i dati, visualizzarli in modo chiaro, eseguire query complesse e ottimizzate, nonché monitorare e ottimizzare le prestazioni del database. Questa potente interfaccia semplifica notevolmente il processo di sviluppo e gestione dei database MongoDB, contribuendo a migliorare l'efficienza e la produttività degli utenti nell'ecosistema di MongoDB.

3.3 Ganache

Per la simulazione della rete è stato utilizzato Ganache [32], uno strumento di sviluppo blockchain che offre un ambiente locale per eseguire test e simulazioni di transazioni su una rete Ethereum. Si tratta di una delle scelte più comuni per gli sviluppatori di applicazioni decentralizzate (DApp) per creare e testare le proprie applicazioni in un ambiente controllato e privato prima di implementarle sulla rete Ethereum pubblica. Ganache, perciò, crea un ambiente di sviluppo "sandbox" in cui gli sviluppatori possono eseguire test di transazioni senza alcun costo. I vantaggi di utilizzare Ganache per le prove di transazioni Ethereum sono molteplici:

1. Ambiente di sviluppo controllato: Ganache consente di avere il controllo completo sull'ambiente di test; è infatti posibile creare account utente predefiniti con un saldo iniziale specifico, generare blocchi di transazioni e ripristinare lo stato della blockchain. Questo permette di simulare scenari specifici e testare le applicazioni in un ambiente controllato e riproducibile.

2. Velocità e scalabilità: A differenza delle reti Ethereum pubbliche, Ganache esegue la blockchain localmente sul computer, il che rende le transazioni e la navigazione della blockchain più veloci.

3. Strumenti di debug: Ganache fornisce una serie di strumenti di debugging che semplificano l'analisi e la correzione degli errori.

4. Integrazione con strumenti di sviluppo: Ganache è compatibile con molti strumenti e librerie di sviluppo Ethereum quali Truffle, Remix, MetaMask e altri strumenti per semplificare il tuo flusso di lavoro di sviluppo.

In Figura 3.2 è mostrato uno snapshot dell'applicazione in cui si possono notare gli indirizzi con i rispettivi bilanci di Ether, il numero di transazioni inviate da ognuno di essi ed altre informazioni.

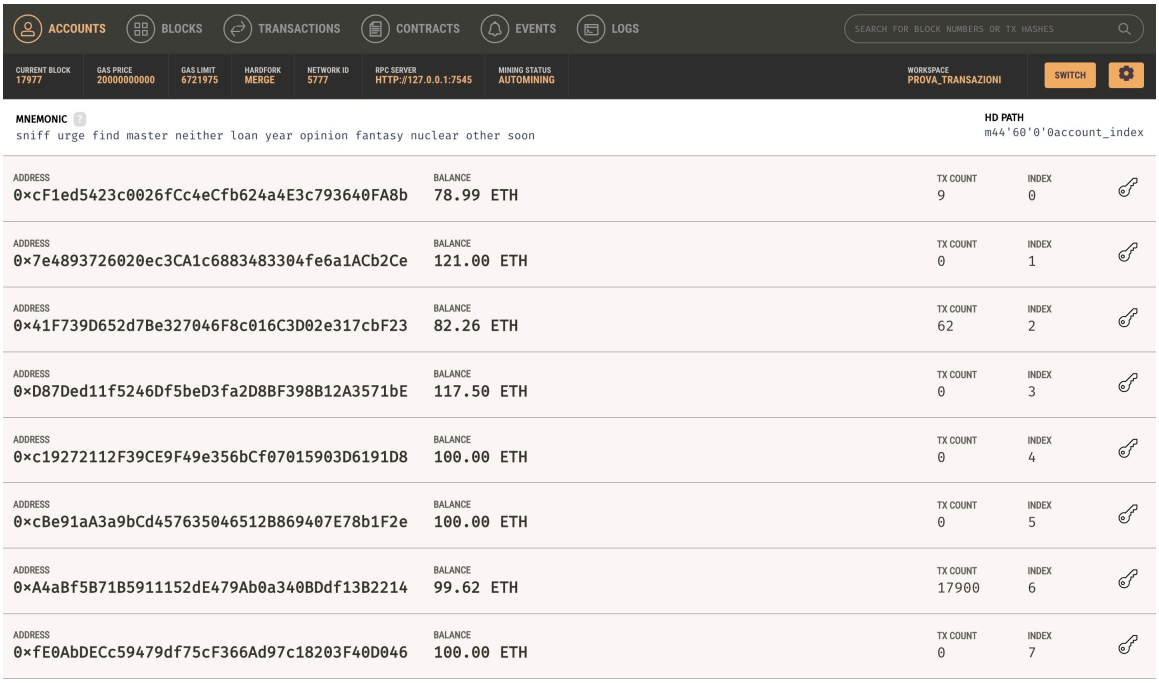

Figura 3.2: Ganache

Capitolo 4

Architettura basata su transazioni singole

In questo capitolo viene presentata l'organizzazione del database utilizzato e viene descritta la prima soluzione proposta per la certificazione dei dati. Tale proposta si basa sull' invio di una transazione Ethereum per ogni elemento contenuto nel database andando ad inserire nel campo "data" di una transazione Ethereum una rappresentazione compressa ed univoca del dato originario; ciò è possibile attraverso l'utilizzo di una funzione hash, descritta in Sezione 4.1.

4.1 Funzione hash

Una funzione hash [33] è una funzione h che ha un input m che può avere lunghezza qualunque e produce un output randomico che ha lunghezza fissa, chiamato $h(m)$. Una prima caratteristica necessaria delle funzioni hash è la suriettività [34]: una funzione viene definita suriettiva quando ogni elemento del codominio è immagine di almeno un elemento del dominio. Il numero di valori che può assumere l'output della funzione è quindi potenzialmente inferiore a quello dei suoi input, come mostrato in Figura 4.1.

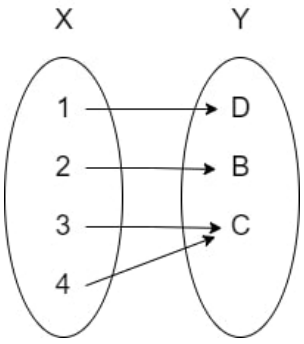

Figura 4.1: Funzione suriettiva [34].

Da questa proprietà deriva intrinsecamente uno dei problemi principali per le funzioni hash, ovvero il problema delle collisioni [35]: una collisione è un evento in cui due esecuzioni della funzione hash producono lo stesso output. Una funzione hash ben progettata deve rendere tali collisioni rare e uniformemente distribuite.

La suriettività non è una condizione sufficiente per le funzioni hash crittografiche; infatti, devono essere verificate altre condizioni.

1. Dato un messaggio m, il suo digest $h(m)$ deve essere calcolato velocemente;
2. Dato un digest y, è computazionalmente intrattabile trovare un messaggio m' tale che $h(m') = y$. Questo significa che $h(m)$ è una funzione unidirezionale (one way), ovvero che è resistente all'inversione.
3. Fissato x, è computazionalmente intrattabile trovare $x' \neq x$ tale che $h(x') = h(x)$. In tal caso la funzione h si dice debolmente resistente alle collisioni. Questo tipo di resistenza viene chiamata debole perché uno dei due input è fissato e quindi l'ipotetico attaccante non è completamente libero di trovare una collisione qualunque ma è vincolato.
4. È computazionalmente intrattabile trovare due messaggi diversi m_1 e m_2, tali che $h(m_1) = h(m_2)$. Questo significa che $h(m)$ è fortemente resistente alle collisioni. In questo caso viene rimosso il vincolo su uno dei due input, perciò l'attaccante si trova in una condizione di massima libertà.

Nel corso degli anni sono stati sviluppati algoritmi in grado di ottenere delle funzioni che rispettassero tali caratteristiche. In Figura 4.2 sono mostrati i principali, insieme alle loro caratteristiche [36].

Algorithm and variant	Output size (bits)	Internal state size (bits)	Block size (bits)	Operations	Collisions found	First published
MD5 (as reference)	128	128	512	And,or,xor, not, Add (mod 2^{32})	Yes	1992
SHA-0	160	160	512	And, Xor, Or, Rot, Add (mod 2^{32})	Yes	1993
SHA-1	160	160	512	And, Xor, Or, Rot, Add (mod 2^{32})	Yes	1995
SHA2-256	256	256	512	And, Xor, Or, Rot, Shr, Add (mod 2^{32})	No	2001
SHA3-256	256	1600	1088	And, Xor, Rot, Not	No	2015

Figura 4.2: Caratteristiche delle funzioni hash note [36].

MD4, MD5 sono due funzioni storiche che oggi non vengono più utilizzate, mentre ebbe molto più successo la famiglia SHA.
SHA (Secure Hash Algorithm) fu sviluppato originariamente dall'NSA (National Security Agency) e pubblicato come standard NIST (National Insistute of Standards

and Technology) nel 1993 come versione SHA-0. Fu però individuata una debolezza che lo fece revisionare nel 1995 diventando SHA-1 con un digest di 160 bit. Nel 2001 sono state aggiunte quattro versioni, raggruppate sotto il nome di SHA-2, che erano caratterizzate sostanzialmente da un digest più lungo.
Due versioni troncate SHA-224 e SHA-348 e due versioni a lunghezza piena SHA-256 e SHA-512.
È importante soffermarsi sul passaggio a SHA-3, poiché con esso cambia il metodo; infatti non viene più utilizzato l'approccio basato sullo schema di Merkle e Damgård tipico del progetto da SHA-0 fino a SHA-2. Tale schema [37] permetteva di ottenere una funzione hash attraverso l'utilizzo iterativo di una funzione di compressione unidirezionale f che ha un ingresso di $n + m$ bit e un'uscita di n bit, come mostrato in Figura 4.3. IV rappresenta il valore iniziale da dare in ingresso insieme al primo blocco in cui è stato diviso il messaggio di cui si vuole calcolare il digest.

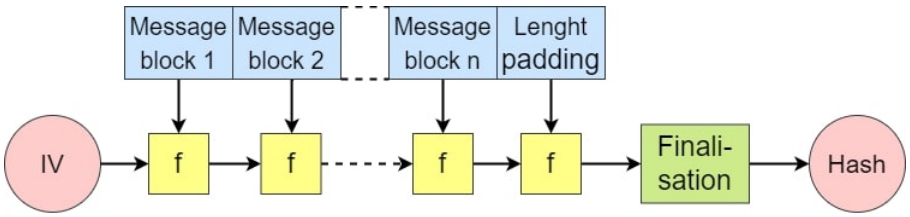

Figura 4.3: Schema Merkle Damgård [37].

A causa di alcune debolezze individuate, come gli attacchi basati su estensione del messaggio e le multicollusioni, il NIST aprì una competizione pubblica in cui l'algoritmo vincitore, Keccak, ha poi composto SHA-3.
Keccak si basa su una costruzione a spugna (sponge construction) [38] in cui si possono individuare due fasi: absorbing e squeezing come mostrato in Figura 4.4.
Il livello di sicurezza è aumentato grazie alla presenza di una variabile di stato interna denominata capacità. Nel presente progetto, l'utilizzo di funzioni hash riveste un ruolo fondamentale, in quanto rappresenta lo strumento mediante il quale si cristallizzano i dati nella blockchain, come verrà spiegato nelle prossime sezioni.

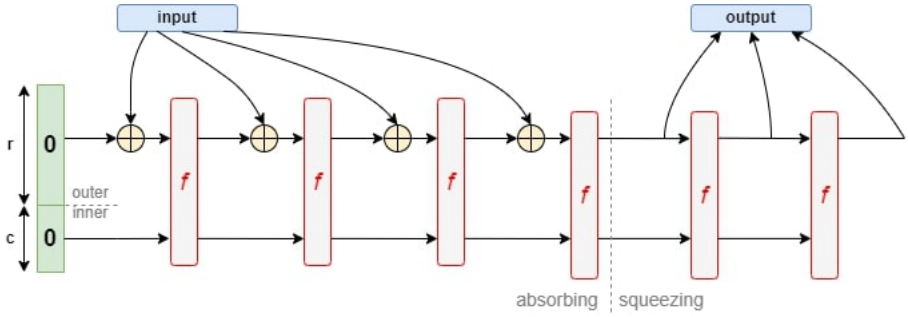

Figura 4.4: Sponge construction [38].

4.2 Organizzazione del Database

Il database utilizzato nel progetto è basato su MongoDB, un sistema di gestione di database orientato ai documenti con cui ci si interfaccia attraverso la libreria JavaScript "Mongoose". Un documento MongoDB, o aggregato, è una struttura dati complessa che raggruppa entità con attributi diversi, non omogenei tra loro. Questo documento è caratterizzato da un livello base di annidamento, rappresentato dalla sua radice, e da ulteriori livelli di annidamento che costituiscono le entità secondarie. La peculiarità di questi documenti risiede nella mancanza di vincoli di omogeneità, consentendo a ciascun elemento di ogni entità di avere attributi diversi. Di conseguenza, solo gli attributi con valori assegnati saranno presenti nel documento.

Ad esempio, gli attributi relativi alle informazioni di certificazione di un contenuto saranno inclusi nel documento solo nel momento in cui questo viene certificato; altrimenti, tali attributi non saranno presenti nel documento. Ciò offre flessibilità nella rappresentazione dei dati, adattandosi dinamicamente alle informazioni specifiche associate a ciascuna entità. Ad ogni azienda corrisponde un aggregato "AZIENDA", un documento MongoDB che tenderà a crescere sempre più in funzione del numero dei dati caricati per la certificazione. Per questo motivo, si è deciso di definire come collezione l'insieme dei documenti MongoDB relativi ad una stessa azienda. Di questa collezione, il primo documento contiene informazioni relative all'azienda quali il nome, l'identificativo dell'azienda e del progetto, l'indirizzo della sede e il numero di telefono, come mostrato in Figura 4.5.


```
_id: ObjectId('654fc146d4b1cb886f3ddae0')
companyID: 1
nome: "Azienda 1"
projectID: 0
▾ indirizzo: Object
    via: "via tfhHz"
    numero: 1
    citta: "xVAOA"
    regione: "HDW1C"
    nazione: "0dvoL"
telefono: "5265403269"
```


Figura 4.5: Documento informativo sulle singole aziende

Invece, in Figura 4.6 è riportato un esempio di documento relativo ai contenuti e agli identificativi della certificazione in blockchain; il capitolo 5 permetterà una piena comprensione di tali campi poiché verranno introdotti i principi cardine della struttura Merkle tree.


```
_id: ObjectId('654fc14dd4b1cb886f3ddaea')
infoID: 0
supplyChainID: "A1_2"
userID: 1
formID: "F1"
hidden: false
loadingTS: 1699725565173
certRequestTS: 1699725805173
▾ data: Object
  ▾ content: Array (4)
    ▾ 0: Object
        formFieldName: "Operazione"
        formFieldValue: "Step 8"
    ▾ 1: Object
        formFieldName: "Descrizione"
        formFieldValue: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque …"
    ▾ 2: Object
        formFieldName: "Data di esecuzione"
        formFieldValue: 1699725601092
    ▾ 3: Object
        formFieldName: "Documento"
        formFieldValue: Binary.createFromBase64('JVBERi0xLjYKJeLjz9MKNyAwIG9iago8PC9TdWJ0eXBlL0Zvc
  sizeKB: 1643.1533203125
▾ blockchainData: Object
  userID: 1
  transactionTS: 1699725822970
  hashRoot: "0x6e510bc5ab5d2af3dcdcad45263359b99d149f7e6af3bb9d7e99e58ac0ea45f2"
  dataHash: "0xba2040b31162dc6be40f89953170b98f5124e0e623aecb1e139d465a2cbaf63c"
  treeDocID: ObjectId('654fc1fe67497f154e2f9518')
  inTreePosition: 7
  transactionHash: "0x660381e2196c06bd02347efdf32104321ed27b402b7239154ee5eef2e994f236"
  blockNumber: 38556
  blockHash: "0x13667e87f86b6ddee52cd33bb53b0321f8793b9ab5c12685295195f1f0e56576"
```


Figura 4.6: Documento sui contenuti

I principali campi per la fase di certificazione e relativa verifica sono i seguenti:

- `loadingTS`: Timestamp di caricamento del documento.
- `certRequestTS`: Timestamp di richiesta di certificazione.
- `data`: Un oggetto che contiene i dati effettivi del contenuto, composto da "content" (array di oggetti) e "sizeKB" (dimensione in kilobyte dei dati).
- `transactionTS`: Timestamp della transazione.

- `hashRoot`: Una stringa che rappresenta la Merkle root, ovvero il dato certificato in chain.
- `treeDocID`: Un puntatore all'albero che ha come radice hashRoot.
- `inTreePosition`: Un identificativo univoco associato alla posizione del dato all'interno dell'albero.
- `transactionHash`: Rappresenta l'hash della transazione.
- `blockNumber`: Il numero del blocco a cui appartiene questa transazione.
- `blockHash`: L'hash del blocco a cui appartiene questa transazione.

Le aziende partecipanti hanno la possibilità di caricare dati attraverso form personalizzati, specifici per il contesto applicativo in cui si trovano. Questi form (Figura 4.7) consentiranno alle aziende di compilare e successivamente inserire i dati in un database dedicato. I campi presenti nei form possono variare a seconda del contesto e rifletteranno diversi aspetti dell'attività aziendale. Inoltre, sarà fornita la possibilità di caricare file di varie estensioni attraverso questi form, consentendo alle aziende di allegare documenti pertinenti al contesto.

Form				
	Doc 001	ID template		
		Nome		
		Breve descrizione		
		Versione		
		Campi Form (lista)	1	Nome campo
				Tipo campo [One-choice|Multi-choice|Txt|binary|Timestamp]
				Valori opzionali
			2	...
			3	...
			...	...
	Doc 002	...		

Figura 4.7: Form per il caricamento dei dati

Nella simulazione svolta il numero delle Aziende e dei relativi documenti possono essere settati attraverso i parametri “numAziende” e “numContenuti”; ad esempio, nel listing 4.1 sono state create 5 aziende in cui vengono suddivisi 1000 contenuti. Il resto del codice contiene le funzioni per la connessione al database, la lettura dei file e il popolamento del database attraverso la generazione dei contenuti.

```
1
2  const mongoose = require('mongoose');
3  const fs = require('fs');
4  const util = require('util');
5  const readFileAsync = util.promisify(fs.readFile);
6
7  const randomstring = require('randomstring');
8
9  const mongoURI = 'mongodb://localhost:27017/dbTest1'
10
11
12 async function connectToDbMongo() {
13     mongoose.set('strictQuery',false);
14
15     await mongoose.connect(mongoURI, {
16         bufferCommands: false,
17         autoCreate: false
18     })
19         .catch(err => console.log(err));
20
21     return mongoose.connection;
22
23 }
24
25
26
27 async function readFile(path) {
28     try {
29         return await readFileAsync(path);
30     } catch (error) {
31         console.error('Errore durante la lettura del file:', error);
32     }
33 }
34
35 async function generaCollezioni(numAziende){
36     const documents = [];
37     for (let i=1; i<=numAziende; i++){
38         const doc = {
39             companyID: i,
40             nome: 'Azienda ' + i.toString(),
41             projectID: 0,
42             indirizzo:{
43                 via: 'via ' + randomstring.generate(5),
44                 numero: i,
45                 citta: randomstring.generate(5),
46                 regione: randomstring.generate(5),
47                 nazione: randomstring.generate(5)
48             },
49             telefono:  Math.floor(Math.random() * 10000000000).toString(),
50         };
51         documents.push(doc);
```

```
52      }
53      return documents;
54  }
55
56
57  async function generaContenuti(nContenuti, numAziende){
58      function divideObjectsIntoGroups(totalObjects, m, limitPerGroup) {
59          if (m <= 0) {
60              throw new Error('Il numero di gruppi deve essere maggiore di zero.
                    ');
61          }
62
63          if (totalObjects <= 0) {
64              throw new Error('Il numero totale di oggetti deve essere maggiore
                    di zero.');
65          }
66
67          if (limitPerGroup <= 0) {
68              throw new Error('Il limite per gruppo deve essere maggiore di zero
                    .');
69          }
70
71          const numberOfGroups = Math.min(m, Math.ceil(totalObjects /
                limitPerGroup));
72          const groups = Array.from({ length: numberOfGroups }, () => 0);
73
74          for (let i = 0; i < totalObjects; i++) {
75              const currentGroupIndex = i % numberOfGroups;
76              groups[currentGroupIndex]++;
77              if (groups[currentGroupIndex] === limitPerGroup) {
78                  const nextGroupIndex = (currentGroupIndex + 1) %
                        numberOfGroups;
79                  [groups[currentGroupIndex], groups[nextGroupIndex]] = [groups[
                        nextGroupIndex], groups[currentGroupIndex]];
80              }
81          }
82          return groups;
83      }
84
85      async function getRandomData(){
86          let forms = ['F0', 'F1'];
87          const choosen = forms[Math.floor(Math.random() * forms.length)];
88          let d;
89
90          const currentTime = new Date().getTime();
91          const randomTimeOffset = Math.floor(Math.random() * 100000);
92          const time = currentTime - randomTimeOffset;
93
94          if (choosen === forms[0]){
95              const img = await readFile('./vigna.jpeg')
96              d = [
97                  {
98                      formFieldName: "Operazione",
99                      formFieldValue: "Step " + Math.floor(Math.random() * 10)
100                 },
101                 {
102                     formFieldName: "Descrizione",
103                     formFieldValue: "Lorem ipsum dolor sit amet, consectetur
                            adipiscing elit. Pellentesque facilisis tellus in quam
```

```
                        tincidunt sodales. In vel velit ac libero tempor
                       lacinia ut ac metus. Vestibulum porta tortor eget nisi
                        pulvinar molestie."
104                 },
105                 {
106                     formFieldName: "Data di esecuzione",
107                     formFieldValue: time
108                 },
109                 {
110                     formFieldName: "Immagine",
111                     formFieldValue: img
112                 }
113             ]
114         }
115
116         if (choosen === forms[1]) {
117             const doc = await readFile('./sample.pdf')
118             d = [
119                 {
120                     formFieldName: "Operazione",
121                     formFieldValue: "Step " + Math.floor(Math.random() * 10)
122                 },
123                 {
124                     formFieldName: "Descrizione",
125                     formFieldValue: "Lorem ipsum dolor sit amet, consectetur
                        adipiscing elit. Pellentesque facilisis tellus in quam
                        tincidunt sodales. In vel velit ac libero tempor
                        lacinia ut ac metus. Vestibulum porta tortor eget nisi
                        pulvinar molestie."
126                 },
127                 {
128                     formFieldName: "Data di esecuzione",
129                     formFieldValue: time
130                 },
131                 {
132                     formFieldName: "Documento",
133                     formFieldValue: doc
134                 }
135             ]
136         }
137         return [d, choosen];
138     }
139
140     const limitePerGruppo = Math.floor(nContenuti/numAziende);
141     const gruppi = divideObjectsIntoGroups(nContenuti, numAziende,
            limitePerGruppo);
142     let contenuti = [];
143     let counterID = 0;
144     let pos = 1;
145     let min = pos;
146     for (const limite of gruppi){
147         let max = min + 5;
148
149         let lim = limite;
150         let tempArray = [];
151         while (lim>0){
152             const currentTime = new Date().getTime();
153             const randomTimeOffset = Math.floor(Math.random() * 100000);
154             const timestampCaricamento = currentTime - randomTimeOffset;
```

```
155             const timestampRichiestaCertificazione = timestampCaricamento + 4
                    * 60000 // 4 minuti avanti
156             //const data =   await readFile('./sample.pdf');
157             let [data, formID] = await getRandomData();
158             let size = JSON.stringify(data).length;
159             const doc = {
160                 infoID: counterID,
161                 supplyChainID: 'A' + pos + '_' + Math.floor(Math.random()*10),
162                 userID: Math.floor(Math.random() * (max - min) + min),
163                 formID: formID,
164                 hidden: Math.random() < 0.3,
165                 loadingTS: timestampCaricamento,
166                 certRequestTS: timestampRichiestaCertificazione,
167                 data: {
168                     content: data,
169                     sizeKB: size/1024
170                 },
171             }
172             tempArray.push(doc);
173             lim--;// = lim-1;
174             counterID = counterID +1;
175         }
176         contenuti.push(tempArray);
177         counterID = 0;
178         pos = pos +1;
179         min = max +1;
180     }
181     return contenuti;
182 }
183
184 // Funzione per inserire i documenti nella collezione "Compagnia"
185 async function insertDocumentsInfo(documents) {
186     let connection = null;
187     try {
188         connection = await connectToDbMongo();
189         for(let i=0; i<documents.length; i++) {
190             let j = i +1
191             let collection = await connection.collection('Azienda' + j);
192             await collection.insertOne(documents[i]);
193         }
194         console.log(`Collezioni create e documenti info inseriti.`);
195     } catch (error) {
196         console.error(`Si  verificato un errore durante l'inserimento dei dati
                : ${error}`);
197     } finally {
198         if (connection){
199             connection.close();
200         }
201     }
202 }
203
204 async function insertDocumentsContenuti(contenuti) {
205     let connection = null;
206     try {
207         connection = await connectToDbMongo();
208
209         for (let n=0; n<contenuti.length; n++){
210             let j = n+1;
211             let collection = await connection.collection('Azienda' + j);
```

```
212             await collection.insertMany(contenuti[n]);
213         }
214         console.log(`Documenti inseriti nella collezione.`);
215     } catch (error) {
216         console.error(`Si è verificato un errore durante l'inserimento dei
                dati: \${error}`);
217     } finally {
218         if (connection){
219             connection.close();
220         }
221     }
222 }
223
224 async function dropCollections() {
225     let connection=null;
226     try {
227         connection = await connectToDbMongo();
228         const collections = await connection.db.listCollections().toArray();
229         for (let i=0; i<collections.length; i++){
230             let name = collections[i].name;
231             await connection.db.dropCollection(name);
232         }
233
234
235         console.log("Tutti i documenti del database sono stati rimossi.");
236     } catch (error) {
237         console.error(`Si è verificato un errore durante la rimozione dei dati
                : ${error}`);
238     } finally {
239         if (connection){
240             connection.close();
241         }
242     }
243 }
244
245
246 async function main(){
247
248     const numAziende = 5;
249     const numContenuti = 1000; //1000;
250
251     console.log('Rimuovo tutte le collezioni');
252     await dropCollections();
253
254
255     console.log('Genero le collezioni e inserisco in ognuna il documento info'
            );
256     const documentsZero = await generaCollezioni(numAziende)
257     await insertDocumentsInfo(documentsZero);
258
259     console.log('Genero e inserisco i documenti per le aziende');
260     const documentsContenuti = await generaContenuti(numContenuti, numAziende)
            ;
261     await insertDocumentsContenuti(documentsContenuti)
262 }
263
264 main();
```

Listing 4.1: createMongoDocuments.js

4.3 Certificazione basata su singole Transazioni

Avendo definito la struttura del database è possibile introdurre il progetto principale, ovvero lo studio di una modalità di certificazione in grado di sfruttare la caratteristica di immutabilità della tecnologia blockchain. Il primo dei due schemi proposti, nonché il più semplice ed intuitivo, prevede l'invio di una transazione Ethereum, contenente l'hash del dato, per ogni elemento presente nel database. Prima di tutto è necessario stabilire una connessione con il database e con la rete Ethereum locale configurata tramite Ganache; ciò viene svolto dalle funzioni presenti nel listing4.2.

```
1  const mongoURI = 'mongodb://localhost:27017/dbTest1';
2  async function connectToDbMongo() {
3      try {
4          const connection = await mongoose.connect(mongoURI, { useNewUrlParser:
               true, useUnifiedTopology: true });
5          console.log('Connessione al database riuscita.');
6          return connection; // Ritorna l'oggetto di connessione
7      } catch (error) {
8          console.error('Errore durante la connessione al database:', error);
9          process.exit(1);
10     }
11 }
12
13 const web3 = new Web3(ganache_URL);
14
15 const senderAddress = '0xA4aBf5B71B5911152dE479Ab0a340BDdf13B2214';
16 const privateKey = '0
       xb279660602d50b3f1c3ea59c5e560dc30cf01a68bc0598b339f178c35fa1fa10';
17 const receiverAddress = '0xE313852cfb7b61C949DCb1776A1CbadBC10cE269';
18
19 // Verifica che gli indirizzi e le chiavi private siano forniti
20 if (!senderAddress || !privateKey || !receiverAddress) {
21     console.error('Indirizzi o chiavi private mancanti.');
22     process.exit(1); // Esci dallo script in caso di mancanza di
           configurazione
23 }
```

Listing 4.2: Connessione a MongoDB e a Ganache

In aggiunta sono state introdotte alcune funzioni che verranno utilizzate nella funzione principale di certificazione:

- `sendTransactionWithData`: gestisce l'invio di transazioni Ethereum contenenti nel campo "data" l'hash del dato che si vuole certificare, firma la transazione con la chiave privata specificata e restituisce la ricevuta della transazione che contiene informazioni sulla transazione confermata, come l'hash della transazione e il numero di blocco. Questo processo rende ogni transazione unica e identificabile sulla blockchain. Ogni dato estratto dal database viene quindi associato ad una transazione, rappresentando un'azione di registrazione sulla

blockchain. Questa metodologia di invio delle transazioni offre un alto grado di tracciabilità e immutabilità, poiché ogni dato viene registrato come parte di una transazione separata sulla blockchain Ethereum. Ciò è mostrato nel listing 4.3.

```
async function sendTransactionWithData(hash, nonce) {
    try {
        const tx = {
            nonce: nonce,
            to: receiverAddress,
            value: web3.utils.toWei('0', 'ether'),
            gas: 2000000,
            gasPrice: web3.utils.toWei('1', 'gwei'),
            data: hash
        };

        const signed_tx = await web3.eth.accounts.signTransaction(tx,
            privateKey);
        const tx_receipt = await web3.eth.sendSignedTransaction(
            signed_tx.rawTransaction);

        return tx_receipt;
    } catch (error) {
        console.error('Errore durante l'invio della transazione:', error
            );
        throw error;
    }
}
```

Listing 4.3: function sendTransactionWithData

- `fetchDataFromDatabase`: recupera dati specifici dal database MongoDB, in particolare, come è mostrato nello script 4.4:
  - Viene inizializzato un array vuoto dataWithLoadingTS che conterrà i dati recuperati insieme ai relativi timestamp di caricamento.
  - Viene eseguita un'iterazione attraverso tutte le aziende.
  - Viene definito un filtro di query per selezionare i record che devono essere recuperati. In questo caso, si cercano i record con i campi relativi alle informazioni in blockchain vuoti, ovvero quelli che non sono stati ancora certificati.
  - Per ciascun record, vengono estratti i dati desiderati (l'hash del dato e il timestamp di caricamento) e aggiunti all'array risultante dataWithLoadingTS.

```
async function fetchDataFromDatabase(db, numAziende) {
    try {
        const dataWithLoadingTS = [];

```

```
5         for (let i = 1; i <= numAziende; i++) {
6             const collectionName = `Azienda${i}`;
7             const collection = db.collection(collectionName);
8
9             const query = {
10                'blockchainData.transactionTS': 0,
11                'blockchainData.blockNumber': 0,
12                'blockchainData.transactionHash': '',
13                'blockchainData.blockHash':'',
14            };
15
16            const projection = {
17                'blockchainData.dataHash': 1,
18                'loadingTS': 1,
19            };
20
21            const newRecords = await collection.find(query, projection).
                  toArray();
22
23            newRecords.forEach((record) => {
24                dataWithLoadingTS.push({
25                    dataHash: record.blockchainData.dataHash,
26                    loadingTS: record.loadingTS,
27                });
28            });
29        }
30
31        return dataWithLoadingTS;
32    } catch (error) {
33        console.error(`Si è verificato un errore durante il recupero dei
              dati: ${error}`);
34        return [];
35    }
36 }
```

Listing 4.4: function fetchDataFromDatabase

- `updateDatabaseWithBlockInfo`: ha lo scopo di aggiornare record specifici, individuati attraverso i campo "dataHash" e "loadingTS". Viene definito un oggetto di aggiornamento che specifica i nuovi valori dei campi della blockchain, quali il timestamp di certificazione, l'hash della transazione, il numero e l'hash del blocco come mostrato nel listing 4.5.

```
1  async function updateDatabaseWithBlockInfo(db, dataHash,
       timestampCertificazione, blockNumber, transactionHash, loadingTS,
       blockHash) {
2      try {
3          const numAziende = 5; // Sostituisci con il numero reale di
               aziende
4          for (let i = 1; i <= numAziende; i++) {
5              const collection = db.collection(`Azienda${i}`);
6
7              // Costruisci il filtro per selezionare il record corrente
                   con 'Blockchain.certTS' uguale a 0 e corrispondente al
                   dataHash e loadingTS
8              const filter = {
```

```
9                 'blockchainData.transactionTS': 0,
10                'blockchainData.blockNumber': 0,
11                'blockchainData.transactionHash': '',
12                'blockchainData.dataHash': dataHash,
13                'loadingTS': loadingTS,
14            };
15
16            // Costruisci l'oggetto di aggiornamento per il record nel
                  filtro corrente
17            const update = {
18                $set: {
19                    'blockchainData.transactionTS':
                          timestampCertificazione,
20                    'blockchainData.blockNumber': blockNumber,
21                    'blockchainData.transactionHash': transactionHash,
22                    'blockchainData.blockHash':blockHash,
23                },
24            };
25
26            const options = { multi: false }; // Aggiorna solo il primo
                  record trovato nel filtro corrente
27            const result = await collection.updateOne(filter, update,
                  options);
28        }
29    } catch (error) {
30        console.error(`Errore durante l\'aggiornamento del database: ${
              error}`);
31    }
32 }
```

Listing 4.5: function updateDatabaseWithBlockInfo

Nella funzione principale, mostrata nel codice 4.6, viene effettuata la certificazione dei dati iniziali attraverso la creazione di una transazione per ciascun dato presente. L'hash del singolo dato viene incluso nel campo "data" della transazione, permettendo di cristallizzare l'informazione in modo immutabile sulla blockchain. I parametri relativi a ciascuna transazione vengono memorizzati in campi specifici e saranno utilizzati nella fase di verifica per individuare in modo univoco ogni transazione all'interno della blockchain. Sfruttando la libreria "os-utils", vengono raccolti dati significativi per valutare le prestazioni dell'approccio, come l'utilizzo della memoria, l'utilizzo della CPU e il tempo di esecuzione.

Infine, viene calcolato il costo totale delle transazioni in ether. È importante sottolineare che il prezzo dell'ether è altamente volatile e può variare notevolmente nel breve e nel lungo termine. Questa volatilità è influenzata da una serie di fattori, tra cui la domanda e l'offerta sul mercato, l'adozione della tecnologia blockchain, le notizie e gli eventi globali, nonché le dinamiche macroeconomiche Ad esempio, a settembre 2023, il prezzo dell'ether oscillava attorno ai 1 500€, come illustrato nella Figura 4.8.

L'utilizzo di questo valore consente di stimare il costo complessivo delle transazioni in euro, fornendo una valutazione finanziaria dell'operazione.

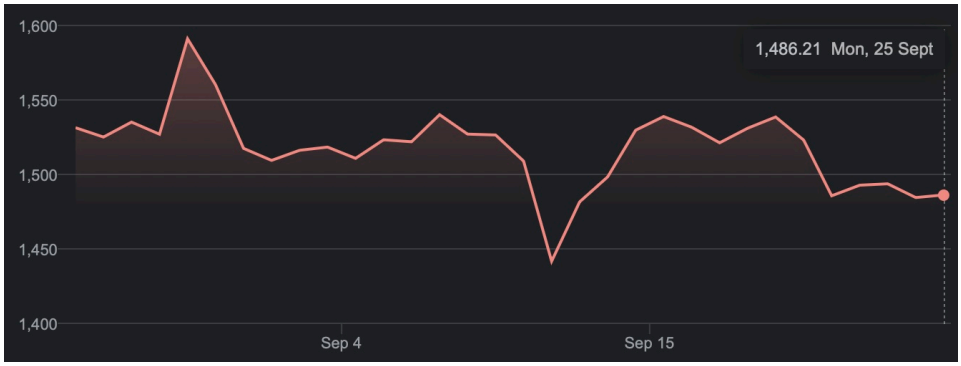

Figura 4.8: Andamento (in €) del prezzo dell'Ether a settembre 2023 [39].

```
1
2  async function processInitialData() {
3      try {
4          // Misura il tempo di esecuzione
5          console.time('processInitialData');
6
7          const connection = await connectToDbMongo();
8          const db = connection.connection.db; // Ottieni l'oggetto db dalla
               connessione
9          let nonce = await web3.eth.getTransactionCount(senderAddress);
10
11         const allDataWithLoadingTS = await fetchDataFromDatabase(db, 5); //
               Passa il numero di aziende
12
13         for (const { dataHash, loadingTS } of allDataWithLoadingTS) {
14             const tx_receipt = await sendTransactionWithData(dataHash, nonce);
15
16             // Calcola il costo della transazione in Ether
17             const gasUsed = tx_receipt.gasUsed;
18             const gasPrice = await web3.eth.getGasPrice();
19             const transactionCostInWei = gasUsed * gasPrice;
20             const transactionCostInEther = web3.utils.fromWei(
                   transactionCostInWei, 'ether');
21
22             const block = await web3.eth.getBlock(tx_receipt.blockNumber);
23             const timestampCertificazione = BigInt(block.timestamp) * 1000n;
                   // Moltiplica per 1000n per ottenere i millisecondi come
                   BigInt
24             const blockNumber = tx_receipt.blockNumber;
25             const transactionHash = tx_receipt.transactionHash;
26             const blockHash = tx_receipt.blockHash;
27
28
29             console.log(`Dato certificato: ${dataHash}`);
30             console.log(`transactionHash: ${transactionHash}`);
31             console.log(`Timestamp di Certificazione: ${
                   timestampCertificazione}`);
32             console.log(`blockNumber: ${blockNumber}`);
33             console.log(`blockHash: ${blockHash}`);
34             console.log(`Costo della transazione Ethereum per un dato: ${
```

```
                transactionCostInEther} Ether`);

            // Aggiorna il database con le nuove informazioni per un dato
            await updateDatabaseWithBlockInfo(db, dataHash,
                timestampCertificazione, blockNumber, transactionHash,
                loadingTS,blockHash);
            nonce++;
        }

        console.timeEnd('processInitialData'); // Termina la misurazione del
            tempo di esecuzione
        osUtils.cpuUsage((cpuUsage) => {
            console.log(`Utilizzo della CPU: ${cpuUsage * 100}%`);
        });

        console.log(`Utilizzo della memoria: ${osUtils.totalmem() - osUtils.
            freemem()} bytes`);
    } catch (error) {
        console.error('Si è verificato un errore durante l\'esecuzione
            iniziale:', error);
    }
}


// Esegui l'elaborazione iniziale dei dati
processInitialData();
```

Listing 4.6: Main function

Infine è stato sviluppato un controllo periodico, eseguito ogni minuto, per monitorare e individuare la presenza di nuovi dati per attuare il processo di certificazione. L'intervallo di tempo tra i controlli è stato impostato a 60 secondi per velocizzare la simulazione; nella realtà, si ipotizza di certificare ogni 24 ore. Durante ogni iterazione, il sistema stabilisce una connessione al database e identifica i nuovi dati registrati da quando è stato effettuato l'ultimo controllo. Questo processo assicura che tutti i dati siano sempre aggiornati e certificati, garantendo un sistema affidabile e conforme alle specifiche esigenze. Tale meccanismo è mostrato nel listing 4.7.

```
setInterval(async () => {
    console.log('Verifica dei nuovi dati nel database...');
    const connection = await connectToDbMongo();
    const db = connection.connection.db; // Ottieni l'oggetto db dalla
        connessione
    let nonce = await web3.eth.getTransactionCount(senderAddress);

    // Recupera il numero di aziende
    const numAziende = 5;

    const allDataWithLoadingTS = await fetchDataFromDatabase(db, numAziende);
        // Passa il numero reale di aziende

    // Verifica se ci sono nuovi dati
    if (allDataWithLoadingTS.length === 0) {
        console.log('Nessun nuovo dato trovato nel database.');
        return; // Esci dalla funzione se non ci sono nuovi dati
    }
```

```
17
18      for (const { datahash, loadingTS } of allDataWithLoadingTS) {
19          const nonceForData = await web3.eth.getTransactionCount(senderAddress)
                ; // Ottieni un nonce separato per ogni dato
20          const tx_receipt = await sendTransactionWithData(datahash,
                nonceForData);
21          const transactionHash = tx_receipt.transactionHash;
22
23          // Calcola il costo della transazione in Ether
24          const gasUsed = tx_receipt.gasUsed;
25          const gasPrice = await web3.eth.getGasPrice();
26          const transactionCostInWei = gasUsed * gasPrice;
27          const transactionCostInEther = web3.utils.fromWei(transactionCostInWei
                , 'ether');
28
29          console.log(`Transazione inviata per un dato. Hash: ${transactionHash
                }`);
30          console.log(`Costo della transazione Ethereum per un dato: ${
                transactionCostInEther} Ether`);
31
32          const block = await web3.eth.getBlock(tx_receipt.blockNumber);
33          const timestampCertificazione = BigInt(block.timestamp) * 1000n;
34          const blockNumber = tx_receipt.blockNumber;
35          const blockHash = tx_receipt.blockHash;
36
37          console.log(`Timestamp di Certificazione per un dato: ${
                timestampCertificazione}`);
38          console.log(`Indirizzo Blocco BC per un dato: ${blockNumber}`);
39
40          // Aggiorna il database con le nuove informazioni per il dato corrente
41          await updateDatabaseWithBlockInfo(db, datahash,
                timestampCertificazione, blockNumber, transactionHash, loadingTS,
                blockHash);
42          nonce++;
43      }
44  }, 60000); // Esegui questa funzione ogni minuto
```

Listing 4.7: Controllo periodico

Un esempio rappresentativo dell'output del processo è evidenziato nella Figura 4.9. Un'analisi approfondita sulle prestazioni e i relativi costi sarà condotta nel Capitolo 6, consentendo un confronto con quelli emersi dalla seconda architettura.

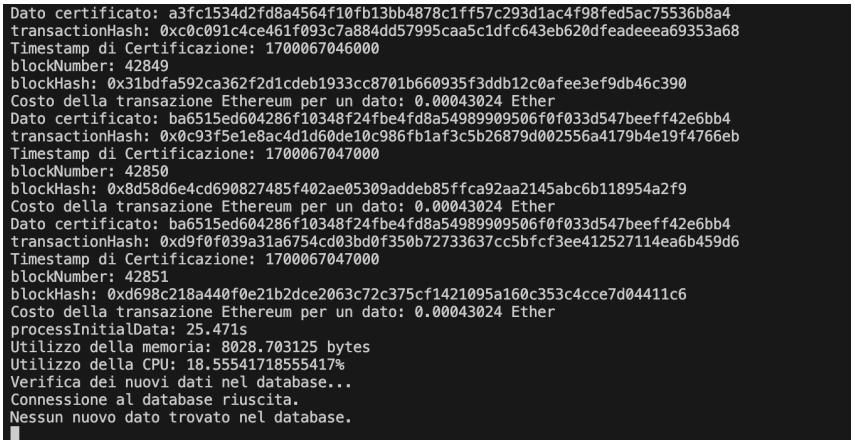

Figura 4.9: Output terminale

4.3.1 Verifica della certificazione

La fase di verifica costituisce un aspetto cruciale per un protocollo di certificazione. Prima di considerare la fase di validazione si presuppone che l'utente, il quale desidera verificare la certificazione di un determinato file, faccia una richiesta dei riferimenti della transazione Ethereum utilizzata per certificare il dato in modo da avere a disposizione tutto ciò di cui necessita per individuare la specifica transazione.

Per questo, la funzione "richiestacertifica" prende in ingresso un file, ne calcola l'hash attraverso "getHashFromFile", lo individua nel database attraverso "findImageInDatabase" e restituisce gli identificativi della transazione ovvero il timestamp di certificazione, l'hash della transazione, il numero e l'hash del blocco nella blockchain.

Attuando questo processo è possibile invocare la funzione di verifica che individua la transazione appropriata attraverso tali parametri, estrae il campo dati ad essa associato e lo confronta con l'hash calcolato localmente dal proprio file. Se i due valori coincidono, la certificazione è confermata come valida, attestando l'integrità del dato. Al contrario, se non corrispondono, la verifica d'integrità fallisce, indicando una possibile alterazione dei dati. Di seguito è riportato il listing 4.8 relativo a questa procedura:

```
1  const crypto = require('crypto');
2  const { default: Web3 } = require('web3');
3  const mongoose = require('mongoose');
4  const fs = require('fs');
5  const util = require('util');
6  const osUtils = require('os-utils'); // Aggiunto per il monitoraggio delle
       risorse
7  const readFileAsync = util.promisify(fs.readFile);
8
9  // Connessione al database MongoDB
10 const mongoURI = 'mongodb://localhost:27017/dbTest1';
11
12 // Funzione per connettersi al database MongoDB
13 async function connectToDbMongo() {
```

```
14      try {
15          const startTime = new Date(); // Misura il tempo di inizio
16          const connection = await mongoose.connect(mongoURI, { useNewUrlParser:
                true, useUnifiedTopology: true });
17          console.log('Connessione al database riuscita.');
18          return connection; // Ritorna l'oggetto di connessione
19      } catch (error) {
20          console.error('Errore durante la connessione al database:', error);
21          process.exit(1);
22      }
23  }
24
25  // Configura la connessione Web3
26  const ganache_URL = 'HTTP://127.0.0.1:7545';
27  const web3 = new Web3(ganache_URL);
28
29  const senderAddress = '0xA4aBf5B71B5911152dE479Ab0a340BDdf13B2214';
30
31  // Verifica che l'indirizzo del mittente sia fornito
32  if (!senderAddress) {
33      console.error('Indirizzo del mittente mancante.');
34      process.exit(1); // Esci dallo script in caso di mancanza di
            configurazione
35  }
36
37  function calculateHash(data) {
38      const hash = crypto.createHash('sha3-256');
39      hash.update(data);
40      return hash.digest('hex');
41  }
42
43  // Funzione per ottenere l'hash dei dati da un file
44  async function getHashFromFile(filePath) {
45      try {
46          const data = await readFileAsync(filePath);
47          return calculateHash(data);
48      } catch (error) {
49          console.error(`Errore durante la lettura del file: ${error}`);
50          return null;
51      }
52  }
53
54  // Funzione per cercare l'immagine nel database
55  async function findImageInDatabase(imageHash, azienda) {
56      try {
57          const connection = await connectToDbMongo();
58          const db = connection.connection.db;
59          const collectionName = `Azienda${azienda}`;
60
61          const query = {
62              'blockchainData.dataHash': imageHash,
63          };
64
65          const projection = {
66              'blockchainData.blockNumber': 1,
67              'blockchainData.transactionHash': 1,
68              'blockchainData.transactionTS': 1,
69              'blockchainData.blockHash':1,
70          };
```

```
71
72          const imageRecord = await db.collection(collectionName).findOne(query,
                projection);
73
74          if (!imageRecord) {
75              console.log('Immagine non trovata nell\'azienda specificata.');
76              return null;
77          }
78
79          const { blockchainData } = imageRecord;
80          const { blockNumber, transactionHash, transactionTS, blockHash} =
                blockchainData;
81
82          return {
83              blockNumber,
84              transactionHash,
85              transactionTS,
86              blockHash,
87          };
88      } catch (error) {
89          console.error('Errore durante la ricerca dell\'immagine nel database:'
                , error);
90          return null;
91      }
92  }
93
94
95
96  // Funzione per cercare la transazione e leggere il campo dati
97  async function getTransactionDataFromBlock(transactionHash) {
98      try {
99          const transaction = await web3.eth.getTransaction(transactionHash);
100         if (!transaction) {
101             console.error(`Transazione non trovata per l'hash: ${
                    transactionHash}`);
102             return null;
103         }
104
105         const transactionData = transaction.input;
106
107         return transactionData;
108     } catch (error) {
109         console.error(`Errore durante la ricerca della transazione: ${error}`)
                ;
110         return null;
111     }
112 }
113
114
115 // funzione che richiede certificazione
116 async function richiestacertifica(filePath, azienda) {
117     const calculatedHash = await getHashFromFile(filePath);
118
119     // Trova il dato nel database
120     const DataInfo = await findImageInDatabase(calculatedHash, azienda);
121
122     if (!DataInfo) {
123         console.error('Dato non trovato sulla blockchain.');
124         return null; // Ritorna null o un valore di default in caso di errore
```

```
125     }
126
127     const { blockNumber, transactionHash, transactionTS, blockHash} = DataInfo
            ;
128
129
130     // Restituisci un oggetto contenente tutte e tre le quantità
131     return { blockNumber, transactionHash, transactionTS, blockHash };
132 }
133
134
135 // Funzione principale per la verifica dell'integrità dei dati
136 async function verifyDataIntegrity(filePath, transactionHash) {
137     try {
138         const startTime = new Date(); // Misura il tempo di inizio
139         const calculatedHash = await getHashFromFile(filePath);
140         console.log('Hash calcolato:', calculatedHash);
141         if (!calculatedHash) {
142             console.error('Impossibile calcolare l\'hash per il file
                    specificato.');
143             return;
144         }
145
146         const transactionData = await getTransactionDataFromBlock(
                transactionHash);
147         console.log('transactionData:', transactionData);
148
149         if (!transactionData) {
150             console.error('Dati della transazione non trovati.');
151             return;
152         }
153
154         // Rimuovi il prefisso "0x" da transactionData
155         const cleanedTransactionData = transactionData.startsWith('0x') ?
                transactionData.slice(2) : transactionData;
156
157         if (calculatedHash === cleanedTransactionData) {
158             console.log('Verifica di integrità riuscita: I dati corrispondono
                    alla transazione sulla blockchain.');
159         } else {
160             console.log('Verifica di integrità fallita: I dati non
                    corrispondono alla transazione sulla blockchain.');
161         }
162
163         const endTime = new Date(); // Misura il tempo di fine
164         const executionTime = endTime - startTime; // Calcola il tempo di
                esecuzione in millisecondi
165
166         console.log(`Tempo di esecuzione: ${executionTime} ms`);
167
168         osUtils.cpuUsage((cpuUsage) => {
169             console.log(`Utilizzo della CPU: ${cpuUsage * 100}%`);
170         });
171
172         console.log(`Utilizzo della memoria: ${osUtils.totalmem() - osUtils.
                freemem()} bytes`);
173     } catch (error) {
174         console.error(`Errore durante la verifica dell'integrità dei dati: ${
                error}`);
```

```
    }
}

// Esegui la verifica dell'integrità dei dati per il file specificato
const filePath = 'sample.pdf'; // Sostituisci con il percorso al tuo file

// Definisci una funzione asincrona per avvolgere il tuo codice principale
async function main() {
    // Utilizza la funzione richiestacertifica
    const result = await richiestacertifica(filePath, 1);

    if (result) {
        console.log('Valori restituiti:', result);

        // Passa il valore di transactionHash a verifyDataIntegrity
        await verifyDataIntegrity(filePath, result.transactionHash);
    } else {
        console.error('Errore durante la richiesta di certificazione.');
    }
}

main();
```

Listing 4.8: Procedura di Verifica d'Integrità

L'esecuzione di questa procedura genera un output dettagliato, come illustrato nella Figura 4.10, confermando l'esito positivo della verifica.

```
Connessione al database riuscita.
Valori restituiti: {
  blockNumber: 41857,
  transactionHash: '0x279ec78d703a4a462e66dd3a5f848d353fb78900e49de784bbb5b9785efacd0c',
  transactionTS: 1700067022000,
  blockHash: '0xee311dd3c5dffbc2b15c747bdb50c723a42bad6fbd6838bbbc49423fd1d82dcc'
}
Verifica di integrità riuscita: I dati corrispondono alla transazione sulla blockchain.
Tempo di esecuzione: 9 ms
Utilizzo della memoria: 8126.140625 bytes
Utilizzo della CPU: 2.7465667915106073%
```

Figura 4.10: Output della Verifica

Capitolo 5

Architettura basata su Merkle tree

Questo capitolo introduce la seconda soluzione proposta per la certificazione dei dati, basata sull'organizzazione dei dati in una struttura chiamata Merkle tree.

5.1 Merkle tree

Il Merkle tree, ideato da Ralph Merkle nel 1979, è una struttura ad albero binario in cui ogni nodo interno rappresenta l'hash crittografico (come ad esempio l'hash SHA-256) dei suoi due figli. Gli hash vengono calcolati ricorsivamente partendo dalle foglie dell'albero e risalendo verso la radice. Questa struttura è costruita a partire da un insieme di dati (come blocchi di transazioni, immagini o altre informazioni) che vengono divisi in blocchi di dimensioni fisse. Di ogni blocco viene poi calcolato il digest per costituire una foglia dell'albero.

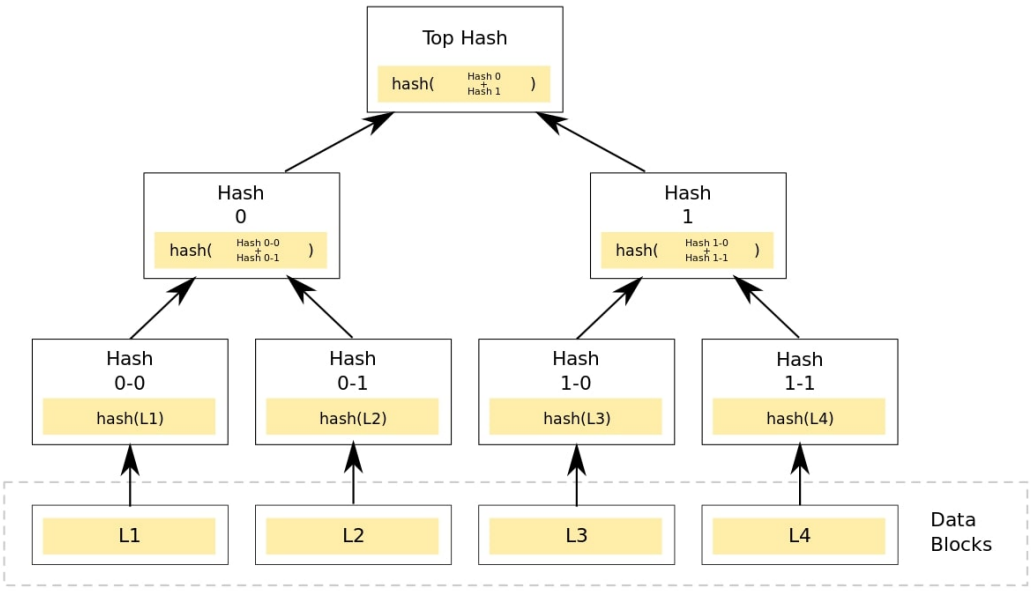

Figura 5.1: Merkle tree [40].

Successivamente, i nodi interni dell'albero vengono costruiti calcolando l'hash delle coppie di nodi figli adiacenti. Questo processo viene ripetuto ricorsivamente fino a raggiungere la radice dell'albero, che rappresenta un unico hash noto come "Merkle Root".
Il Merkle tree possiede proprietà cruciali:

1. `Integrità`: Ogni modifica ai dati in una foglia comporta una modifica nella radice del Merkle tree.

2. `Efficienza`: Per verificare l'integrità di un dato specifico, è sufficiente avere l'hash della radice e una "Merkle proof", ovvero una serie di hash che collega il dato alla radice, il cui numero è proporzionale al logaritmo del numero di foglie dell'albero.

In Figura 5.1 è mostrato un semplice esempio di Merkle tree: ad ognuno dei quattro blocchi di dati (L1, L2, L3, L4) viene associata una foglia dell'albero che ha come etichetta il digest hash del blocco corrispondente. Questi digest vengono, a loro volta, raggruppati. Supponendo che l'albero sia binario, i primi due hash digest vengono combinati nel calcolare un ulteriore hash digest, che rappresenta la somma binaria dei due hash digest precedenti. Lo stesso viene fatto con L3 e L4. Successivamente, i due nodi di livello superiore ottenuti vengono raggruppati in un unico digest che, questa volta, diventa la radice dell'albero.

Prendendo in considerazione un albero con N foglie, in cui N è una potenza di 2, si ha che:

- la dimensione della proof è di $\log_2(N)$ hash.
- Il calcolo della proof ha costo $\mathcal{O}(\log_2(N))$.

In generale quando N non è esprimibile come una potenza di 2 esistono diverse soluzioni e la scelta è lasciata all'implementatore, secondo un trade-off tra complessità e compattezza.

5.2 Certificazione basata su Merkle tree

Nel contesto di questo progetto, l'utilizzo del Merkle tree rappresenta un'innovativa soluzione per organizzare gli hash dei documenti. L'obiettivo primario di questo approccio è limitare il numero di transazioni richieste sulla piattaforma Ethereum, rendendo il processo di certificazione più efficiente ed economicamente vantaggioso. Rispetto all'approccio analizzato in Sezione 4.3, che richiedeva una transazione separata per ciascun dato, il Merkle tree consente di certificare più dati in una singola transazione. Questa transazione contiene la radice del Merkle tree, consentendo così un notevole risparmio in termini di costi e tempo di elaborazione.

L'idea è quella di considerare in modo indipendente i dati delle varie aziende e organizzarli in alberi, la cui numerosità (ovvero, il numero di elementi con i quali

viene costruito l'albero) è una scelta progettuale che tiene conto di vari fattori volti a minimizzare una funzione costo. Un'analisi approfondita di questa tematica verrà affrontata nel Capitolo 6, considerando anche vari scenari applicativi.

Nel progetto, si è optato per la conservazione nel database dell'intera struttura del Merkle tree anziché il salvataggio della Merkle proof per ogni dato da certificare. Questa scelta è stata motivata dalla volontà di evitare l'occupazione inutile di spazio nel database attraverso lo storage di ridondanza associato alla memorizzazione delle Merkle proof per ciascun dato. Il salvataggio dell'albero consente di individuare velocemente i nodi che fanno parte della proof e ciò velocizza il calcolo locale della root durante la fase di verifica. Le funzioni implementate per interagire con il database e la blockchain e per l'invio della transazione rimangono invariate rispetto all'architettura presentata nel Capitolo 4.

Successivamente, sono state implementate due funzioni che introducono funzionalità aggiuntive:

1. La funzione "`updateDatabaseWithBlockInfo`" aggiorna, oltre alle informazioni relative alla transazione in chain anche i campi:
   - `treeDocID`: contiene l'identificativo dell'oggetto in cui è salvato l'albero di appartenenza del dato.
   - `inTreePosition`: contiene l'indice della posizione del dato all'interno dell'albero

   Questi due parametri rivestono un ruolo fondamentale nel processo di verifica in quanto permettono di accedere velocemente all'albero e alla posizione esatta del dato.

2. La funzione "`fetchDataFromDatabase`" introduce la logica di ordinamento dei dati all'interno degli alberi organizzandoli in base all'ordine crescente dei timestamp delle richieste di certificazione.

Le modifiche chiave coinvolgono la funzione di creazione del Merkle tree e la procedura successiva per il suo inserimento nel database. Ora, l'albero generato viene inserito in una nuova collezione, in cui ognuno è identificato univocamente mediante il relativo ObjectID. Le funzioni che permettono ciò sono mostrate nel listing 5.1.

```
1  function buildMerkleTree(hashedData) {
2      if (hashedData.length === 0) {
3          return [null, null];  // Se la lista è vuota, restituisci [null, null]
4      }
5
6      let numLeaves = hashedData.length;
7      let nextPowerOfTwo = 1;
8      while (nextPowerOfTwo < numLeaves) {
9          nextPowerOfTwo *= 2;
10     }
11
```

```
12      // Aggiungi foglie fantasma se il numero di foglie non è una potenza di
            due
13      if (numLeaves !== nextPowerOfTwo) {
14          const numPhantomLeaves = nextPowerOfTwo - numLeaves;
15          const phantomLeaves = new Array(numPhantomLeaves).fill('nodo
                riempitivo'); // Puoi usare un valore specifico per le foglie
                fantasma
16
17          hashedData = hashedData.concat(phantomLeaves);
18      }
19
20      const tree = [hashedData];
21
22      while (tree[tree.length - 1].length > 1) {
23          const level = [];
24          for (let i = 0; i < tree[tree.length - 1].length; i += 2) {
25              const left = tree[tree.length - 1][i];
26              const right = tree[tree.length - 1][i + 1];
27              level.push(crypto.createHash('sha3-256').update(left + right).
                    digest('hex'));
28          }
29
30          tree.push(level);
31      }
32
33      return [tree, tree[tree.length - 1][0]];
34  }
35
36  // Funzione per aggiungere un albero di Merkle alla collezione nel database
37  async function addMerkleTreeToDatabase(db, merkleTree) {
38      try {
39          const collection = db.collection('Trees');
40
41          // Inserisci il nuovo documento con l'albero di Merkle nella
                collezione
42          const result = await collection.insertOne({ merkleTree });
43
44          console.log('Albero di Merkle aggiunto al database con successo.');
45
46          // Restituisci l'ObjectId del documento appena inserito
47          return result.insertedId;
48      } catch (error) {
49          console.error('Errore durante l\'aggiunta dell\'albero di Merkle al
                database:', error);
50          throw error;
51      }
52  }
```

Listing 5.1: Costruzione e salvataggio di Merkle tree

Anche in questa soluzione la funzione principale, listing 5.2, viene interrogata periodicamente in modo da gestire le nuove entry nel database.

```
1   async function processAndUpdateData() {
2       try {
3           // Misura il tempo di esecuzione
4           console.time('processAndUpdateData');
```

```
5
6           const connection = await connectToDbMongo();
7           const db = connection.connection.db;
8           let nonce = await web3.eth.getTransactionCount(senderAddress);
9
10          const numAziende = 1;
11          const aziendeInfo = {};
12
13          for (let aziendaIndex = 1; aziendaIndex <= numAziende; aziendaIndex++)
                {
14
15              const numeroAlberi = 2;
16
17              const { dataHashes, certRequestTS } = await fetchDataFromDatabase(
                    db, aziendaIndex);
18
19              if (dataHashes.length === 0) {
20                  console.log(`Nessun nuovo dato trovato nel database per l'
                        azienda ${aziendaIndex}.`);
21                  continue;
22              }
23
24              // Inizializza l'oggetto aziendaInfo per questa azienda
25              if (!aziendeInfo[aziendaIndex]) {
26                  aziendeInfo[aziendaIndex] = {
27                      numeroAlberi: numeroAlberi,
28                      L: 0,
29                  };
30              }
31
32              const datiPerAlbero = suddividiDatiInAlberi(dataHashes,
                    numeroAlberi);
33
34              for (let i = 0; i < numeroAlberi; i++) {
35                  const merkleTree = buildMerkleTree(datiPerAlbero[i]);
36                  console.log(`Azienda ${aziendaIndex}, Albero ${i} - Dati: ${
                        datiPerAlbero[i].length}`);
37
38                  aziendeInfo[aziendaIndex].L += datiPerAlbero[i].length;
39                  const X = certRequestTS[aziendeInfo[aziendaIndex].L - 1];
40
41                  const tx_receipt = await sendTransactionWithData(merkleTree
                        [1], nonce);
42
43                  const gasUsed = tx_receipt.gasUsed;
44                  const gasPrice = await web3.eth.getGasPrice();
45                  const transactionCostInWei = gasUsed * gasPrice;
46                  const transactionCostInEther = web3.utils.fromWei(
                        transactionCostInWei, 'ether');
47
48                  const block = await web3.eth.getBlock(tx_receipt.blockNumber);
49                  const timestampCertificazione = BigInt(block.timestamp) * 1000
                        n;
50                  const blockNumber = tx_receipt.blockNumber;
51                  const transactionHash = tx_receipt.transactionHash;
52                  const blockHash = tx_receipt.blockHash;
53
54                  console.log(`Transazione inviata per l'azienda ${aziendaIndex
                        }, albero ${i}. Hash: ${transactionHash}`);
```

```
55                 console.log(`Timestamp di Certificazione per l'azienda ${
                       aziendaIndex}, albero ${i}: ${timestampCertificazione}`);
56                 console.log(`Numero Blocco BC per l'azienda ${aziendaIndex},
                       albero ${i}: ${blockNumber}`);
57                 console.log(`Hash Blocco BC per l'azienda ${aziendaIndex},
                       albero ${i}: ${blockHash}`);
58
59                 console.log(`Costo della transazione Ethereum per l'azienda ${
                       aziendaIndex}, albero ${i}: ${transactionCostInEther}
                       Ether`);
60
61                 const treeDocId = await addMerkleTreeToDatabase(db, merkleTree
                       );
62                 await updateDatabaseWithBlockInfo(db, aziendaIndex,
                       timestampCertificazione, blockNumber, transactionHash,
                       merkleTree[1], X, treeDocId, blockHash);
63                 nonce++;
64             }
65         }
66
67         console.timeEnd('processAndUpdateData'); // Termina la misurazione del
               tempo di esecuzione
68         osUtils.cpuUsage((cpuUsage) => {
69             console.log(`Utilizzo della CPU: ${cpuUsage * 100}%`);
70         });
71
72         console.log(`Utilizzo della memoria: ${osUtils.totalmem() - osUtils.
               freemem()} bytes`);
73     } catch (error) {
74         console.error('Si è verificato un errore durante l'esecuzione:', error
               );
75     }
76 }
77
78
79
80
81 // Esegui l'elaborazione iniziale dei dati
82 processAndUpdateData();
83
84 // Esegui il controllo periodico ogni minuti
85 setInterval(async () => {
86     console.log('Verifica dei nuovi dati nel database...');
87     await processAndUpdateData();
88 }, 60000);
```

Listing 5.2: Main function

L'output generato durante l'esecuzione dello script, mostrato in Figura 5.2, evidenzia il comportamento nel caso di una singola azienda che utilizza 2 Merkle tree; oltre agli identificativi della transazione, sono forniti dettagli sulle prestazioni in termini di tempo di elaborazione e utlizzo di risorse. Questi dettagli offrono un'analisi approfondita delle prestazioni e del risultato dell'esecuzione dello script.

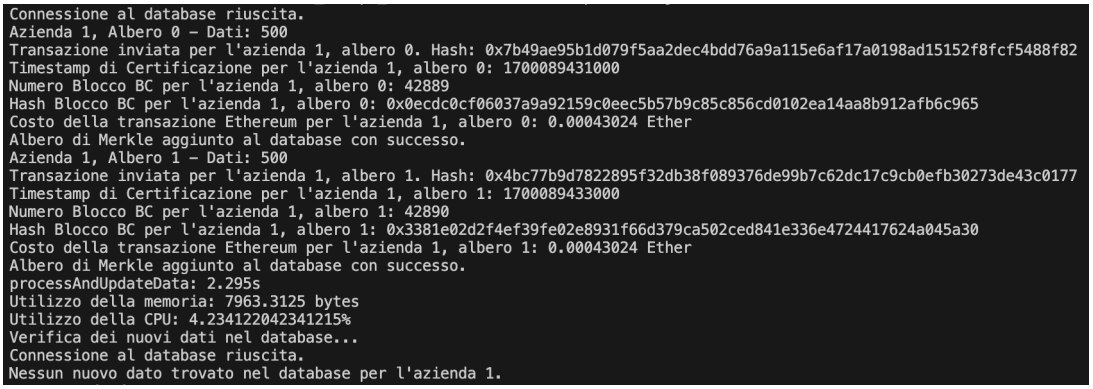

Figura 5.2: Output della fase di certificazione

5.2.1 Verifica della certificazione

La fase di verifica presenta una complessità maggiore rispetto all'approccio presentato in Sezione 4.3. Essa richiede il calcolo locale della Merkle root, partendo dal documento di cui si intende verificare la certificazione e dalla rispettiva proof. La proof viene estratta grazie alla funzione "`getMerkleProofFromDatabase`" (listing 5.3). Questa funzione, dopo aver individuato l'albero corretto e la posizione del dato in questione attraverso i campi "`treeDocID`" e "`inTreePosition`" (output della funzione "richiestacertifica" insieme ai dati relativi alla transazione in chain), chiama la funzione "`getMerkleProof`" per ottenere i nodi con i quali si ripercorre l'albero fino alla radice.

```
1
2  async function getMerkleProofFromDatabase(objectId, index) {
3      try {
4          const connection = await connectToDbMongo();
5          const db = connection.connection.db;
6
7          // Ottieni l'albero dal database usando l'ID dell'oggetto
8          const treeDoc = await db.collection('Trees').findOne({ _id: objectId
               });
9          const tree = treeDoc.merkleTree[0];
10
11         const proof = getMerkleProof(tree, index);
12
13         return proof;
14     } catch (error) {
15         console.error('Errore durante il recupero della prova di Merkle dal
               database:', error);
16         return null;
17     }
18 }
19
20
21
22 function getMerkleProof(tree, index) {
23     const proof = [];
24     let levelIndex = index;
```

```
25
26      for (const level of tree) {
27          if (levelIndex % 2 === 0 && levelIndex + 1 < level.length) {
28              const siblingIndex = levelIndex + 1;
29              proof.push('R' + level[siblingIndex]);
30          } else if (levelIndex % 2 === 1 && levelIndex - 1 >= 0) {
31              const siblingIndex = levelIndex - 1;
32              proof.push('L' + level[siblingIndex]);
33          }
34
35          levelIndex = Math.floor(levelIndex / 2);
36      }
37
38      return proof;
39  }
```

Listing 5.3: getMerkleProofFromDatabase

Quando l'utente dispone della proof e dei dati necessari per individuare la transazione che contiene la root in chain, inizia la funzione di verifica vera e propria che prevede:

1. Calcolo dell'hash del file.
2. Estrazione della root dalla chain.
3. Calcolo locale della root attraverso la proof e l'hash trovato al punto 1.
4. Confronto dei valori ottenuti nei punti 2. e 3.; in caso di uguaglianza, la verifica ha esito positivo.

Il confronto è svolto dalla funzione "verifyMerkleProof", mentre "verifyDataIntegrity" è la funzione principale per la verifica di integrità come mostrato nel listing 5.4.

```
1
2   function verifyMerkleProof(merkleRoot, hashedData, proof) {
3       let hashValue = hashedData;
4
5       for (const proofHash of proof) {
6           if (proofHash.startsWith('L')) {
7               const combinedHash = crypto.createHash('sha3-256').update(
                    proofHash.substring(1) + hashValue).digest('hex');
8               hashValue = combinedHash;
9           } else {
10              const combinedHash = crypto.createHash('sha3-256').update(
                    hashValue + proofHash.substring(1)).digest('hex');
11              hashValue = combinedHash;
12          }
13      }
14
15      // Aggiungi il prefisso "0x" a hashValue se non presente
16      hashValue = hashValue.startsWith('0x') ? hashValue : '0x' + hashValue;
17
18      return hashValue === merkleRoot;
19  }
```

```
20
21
22
23 // Funzione principale per la verifica dell'integrità dei dati
24 async function verifyDataIntegrity(filePath,transactionHash, proof) {
25     try {
26         const startTime = new Date(); // Misura il tempo di inizio
27         const calculatedHash = await getHashFromFile(filePath);
28
29         if (!calculatedHash) {
30             console.error('Impossibile calcolare l\'hash per il file
                   specificato.');
31             return;
32         }
33
34
35         const radicechain = await getTransactionDataFromBlock(transactionHash)
               ;
36
37         if (!radicechain) {
38             console.error('Dati della transazione non trovati.');
39             return;
40         }
41
42         // Verifica la radice di Merkle utilizzando la prova
43         const isMerkleProofValid = verifyMerkleProof(radicechain,
               calculatedHash, proof);
44
45         if (isMerkleProofValid) {
46             console.log('Verifica di integrità riuscita: La Merkle root Ã¨
                   valida.');
47         } else {
48             console.log('Verifica di integrità fallita: La Merkle root non Ã¨
                   valida.');
49         }
50
51         const endTime = new Date(); // Misura il tempo di fine
52         const executionTime = endTime - startTime; // Calcola il tempo di
               esecuzione in millisecondi
53
54         console.log(`Tempo di esecuzione: ${executionTime} ms`);
55
56         osUtils.cpuUsage((cpuUsage) => {
57             console.log(`Utilizzo della CPU: ${cpuUsage * 100}%`);
58         });
59
60         console.log(`Utilizzo della memoria: ${osUtils.totalmem() - osUtils.
               freemem()} bytes`);
61     } catch (error) {
62         console.error('Si Ã¨ verificato un errore durante la verifica di
               integrità:', error);
63     }
64 }
65
66 // Esegui la verifica dell'integrità dei dati per il file specificato
67 const filePath = 'sample.pdf'; // Sostituisci con il percorso al tuo file
68
69 // Definisci una funzione asincrona per avvolgere il tuo codice principale
70 async function main() {
```

```
71      // Utilizza la funzione richiestacertifica
72      const result = await richiestacertifica(filePath, 1);
73
74      if (result) {
75          console.log('Valori restituiti:', result);
76
77          // Passa il valore di transactionHash a verifyDataIntegrity
78          await verifyDataIntegrity(filePath, result.transactionHash, result.
                proof);
79      } else {
80          console.error('Errore durante la richiesta di certificazione.');
81      }
82  }
83
84  // Chiama la funzione principale
85  main();
```

Listing 5.4: Verifica di integrità

Un esempio di verifica è illustrato in Figura 5.3.

```
Connessione al database riuscita.
Valori restituiti: {
  blockNumber: 42889,
  transactionHash: '0x7b49ae95b1d079f5aa2dec4bdd76a9a115e6af17a0198ad15152f8fcf5488f82',
  transactionTS: 1700089431000,
  blockHash: '0x0ecdc0cf06037a9a92159c0eec5b57b9c85c856cd0102ea14aa8b912afb6c965',
  proof: [
    'Rba6515ed604286f10348f24fbe4fd8a54989909506f0f033d547beeff42e6bb4',
    'L8a2f8f40de0b6544976c5987e7ca1cf927d2f2985e75ea4ea065efc641b02858',
    'L0275a1443e465800628f3f8108ace6888d66c2696992b0fa808504fc3a3a84d0',
    'L01a056d0e9971f13f7d6df358acb9b21ef80864e0653b987210e7698afeed62f',
    'R9bf1e5aa5ae5ff39f4d40fc459f4aa91ce7fe3bec693e106f703976d897a60d3',
    'L2ad8b13f36c0dd3672172c20da490679dec80bff30dd481b4a056de9bb0b6614',
    'Leb94bc9bbcc726f5265e8558ef6dfbd6a770a250f03899cf85854af59b89db49',
    'Lf5a473b0f28464921fc6fad998a3d92a85c67f3f4604cc7485a15641cb217216',
    'L50ae0b086e3eae4c2899305f98f376ca6a903b87471058cf5687bf0b7ae1b918'
  ]
}
Verifica di integrità riuscita: La radice di Merkle è valida.
Tempo di esecuzione: 13 ms
Utilizzo della memoria: 8130.421875 bytes
Utilizzo della CPU: 2.4968789013732784%
```

Figura 5.3: Verifica della certificazione con Merkle tree

Capitolo 6

Analisi delle prestazioni e criteri di progetto

Nei Capitoli 4 e 5 sono stati esaminati due approcci per la certificazione dei dati: uno basato su transazioni individuali per ciascun data entry e l'altro basato sulla loro organizzazione in Merkle tree. Nonostante entrambi gli schemi svolgano la stessa funzione, vi sono significative differenze sotto diversi aspetti. Nel presente capitolo, si esamineranno le differenze principali tra questi due approcci, confrontando i risultati emersi dalla simulazione svolta in termini di costo e prestazioni. In seguito, sarà presentata un'analisi generalizzata in cui verrà definita una funzione costo, determinata dalla somma di più contributi, al fine di adattare lo schema a molteplici casi di studio teorici, in grado di modellare situazioni reali.

6.1 Confronto delle prestazioni e i costi

In Tabella 6.1 è mostrata una panoramica, dal punto di vista quantitativo, delle prestazioni dei due approcci al variare della dimensione delle entries da certificare e al numero di alberi costruiti. I parametri presi in considerazione sono:

- CPU usage: indica l'utilizzo della CPU durante il processo di certificazione. I risultati mostrano come, fissata la quantità dei dati da certificare, si ha un utilizzo minore quando non si ricorre ai Merkle tree. Invece, utlizzando tale struttura, il costo aumenta al diminuire della loro numerosità. Questi risultati sono dettati dal fatto che nel caso di utilizzo di Merkle tree incide l'ordinamento dei dati, la costruzione degli gli alberi e il loro salvataggio. Logicamente, all'aumentare dei dati aumenta CPU usage in tutti i casi di studio.

- Storage: esprime, in kilobyte, quanta memoria è necessaria per salvare i Merkle tree. In generale, definendo con N il numero dei dati da certificare e con M il numero dei Merkle tree creati ne segue che è il numero di hash da salvare, definito dalla variabile X, è dato da:

$$X = \left(2 \cdot \frac{N}{M} - 1\right) \cdot M \tag{6.1}$$

Semplificando si ottiene:

$$X = (2 \cdot N - M) \tag{6.2}$$

Perciò, a parità di N, lo storage complessivo è inversamente proporzionale al numero di alberi. In questa simulazione è stato utilizzato l'algoritmo SHA3-256; ne segue che lo storage, espresso in kilobyte, sia dato dall'espressione:

$$Storage = X \cdot 0.032 \tag{6.3}$$

- Execution time: mostra, in secondi, il tempo necessario per tutto il processo di certificazione, partendo dall'invio delle transazioni in blockchain fino all'update del database e il salvataggio degli alberi. Il fattore che incide maggiormente su questo parametro è il numero di transazioni: ciò è evidente considerando che la soluzione con transazioni singole e $N = 128$ ha un tempo maggiore rispetto al caso con $N = 128$ e $M = 1$. Ne segue che, per limitare i tempi di certificazione è necessario considerare una soluzione che invii meno transazioni possibile.
- Transactions Cost: rappresenta il costo in euro delle transazioni Ethereum per ciascun approccio. Il costo è stato calcolato moltiplicando il costo in ether per il valore del tasso di cambio ether/euro a settembre 2023.

Tabella 6.1: Prestazioni della certificazione

Numero di data entries	Numero di Merkle tree	CPU usage	Storage (kB)	Execution Time (s)	Transactions Cost (€)
128	//	30 %	//	5	83.2
128	1	38 %	8.2	0.6	0.65
128	16	36 %	7.7	1.3	10.4
128	64	35 %	6.1	3	41.6
512	//	32%	//	20	332.8
512	1	45%	32.7	0.9	0.6
512	16	41 %	32.3	2.7	10.4
512	64	38 %	30.7	5.7	41.6
1024	//	34 %	//	39	665.6
1024	1	47 %	65.5	2.4	0.6
1024	16	44 %	65.0	4.1	10.4
1024	64	41 %	63.4	7.2	41.6
2048	//	37 %	//	72	1331.2
2048	1	50 %	131.1	4	0.6
2048	16	47 %	130.6	6	10.4
2048	64	43 %	129.02	9	41.6

Di seguito, le figure evidenziano in modo più significativo i risultati ottenuti per i diversi approcci rispetto al numero delle entry del database. Nello specifico, in Figura 6.1 si mostrano i risultati dell'uso della CPU, nella Figura 6.2 sono presentati i risultati relativi allo storage; successivamente, in Figura 6.3 si illustrano i tempi di esecuzione ed infine nella Figura 6.4 sono esposti i costi relativi alle transaioni.

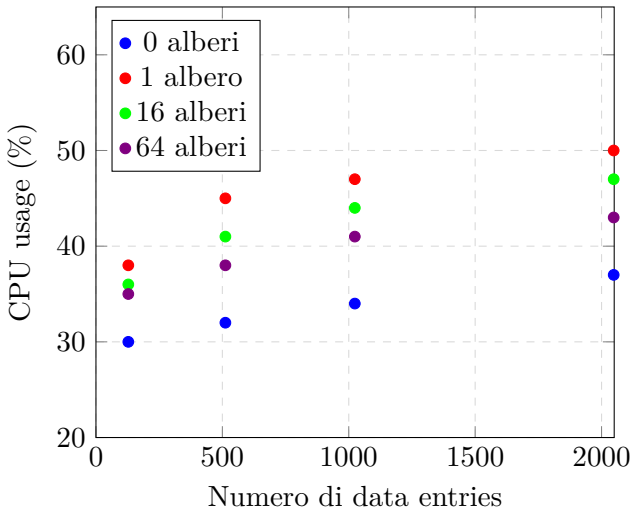

Figura 6.1: Grafico di confronto per CPU usage

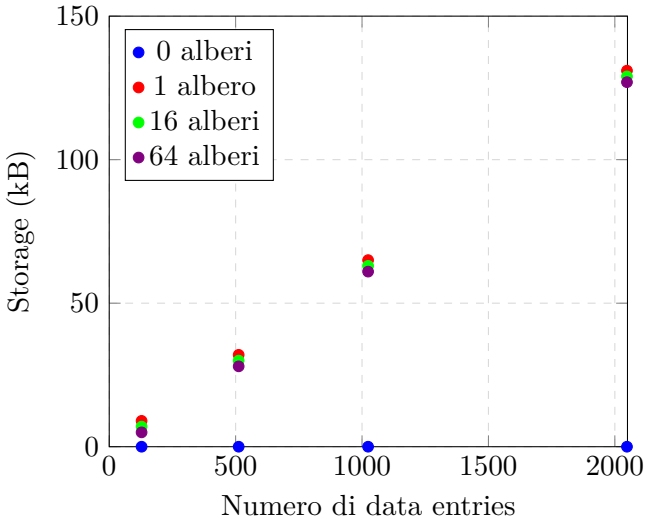

Figura 6.2: Grafico di confronto per lo storage

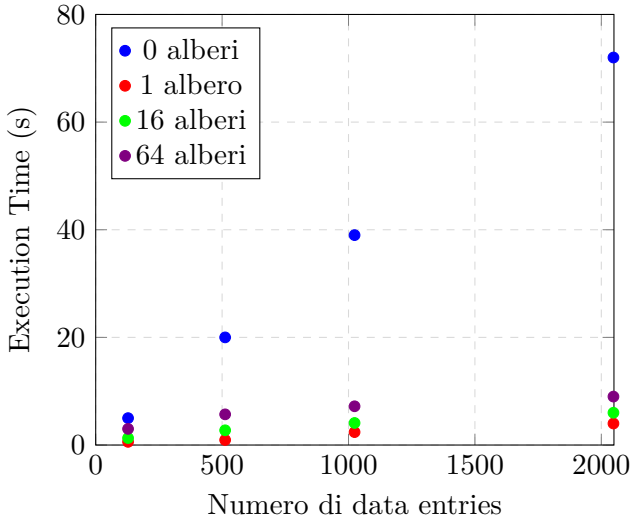

Figura 6.3: Grafico di confronto per Execution Time

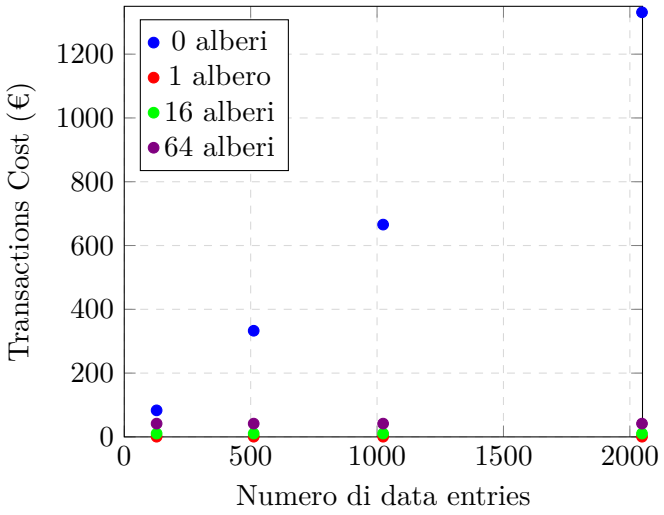

Figura 6.4: Grafico di confronto per Transactions Cost

Infine, la Tabella 6.2 illustra le prestazioni dei due approcci durante la fase di verifica di integrità della certificazione in termini di velocità di esecuzione. Il meccanismo è simile: nell'approccio che non fa uso di alberi si va a confrontare direttamente l'hash del file con il contenuto della transazione Ethereum mentre utilizzando i Merkle tree si ha la necessità di ricalcolare localmente la root a partire dalla proof prima di verificare la sua integrità; questa operazione prevede il calcolo di $\log_2 N'$ hash, dove N' indica il numero di foglie per ogni albero generato ed è definito come $\frac{N}{M}$.

Tabella 6.2: Prestazioni della verifica

Numero di data entries	Numero di Merkle tree	Size Proof	Execution Time (ms)
128	//	//	13
128	1	7	19
128	16	3	16
128	64	1	14
512	//	//	13
512	1	9	20
512	16	5	17
512	64	3	15
1024	//	//	13
1024	1	10	21
1024	16	6	17
1024	64	4	15
2048	//	//	13
2048	1	11	22
2048	16	7	18
2048	64	5	16

I risultati evidenziano la semplicità del processo indipendentemente dalla scelta progettuale, il tempo di esecuzione cresce all'aumentare della dimensione degli alberi considerati, in virtù dell incremento del numero di hash da svolgere per ottenere la proof. Invece, nel primo metodo, la numerosità delle entry non incide sul processo poiché ogni dato viene certificato in modo indipendente dagli altri come mostrato in Figura 6.5.

In conclusione, l'approccio che fa uso dei Merkle tree risulta efficiente anche in fase di verifica, in virtù della velocità nello svolgere le funzioni hash, mantenendo un ottimo bilanciamento tra consumo di risorse e prestazioni considerando l'intero processo. In questa particolare simulazione, è evidente che il termine di maggior impatto in termini di costi sia l'invio delle transazioni in blockchain; perciò, la soluzione ottimale per minimizzare la spesa consiste nella creazione di un unico Merkle tree per azienda.

Questo risultato però non è universale ma dipende dalle diverse esigenze delle aziende. Nel contesto di questa analisi, nella Sezione 6.2, si procede con la generalizzazione della funzione costo incorporando diversi contributi al fine di adattare il modello a una vasta gamma di scenari di studio al fine di individuare l'architettura che minimizza tale funzione.

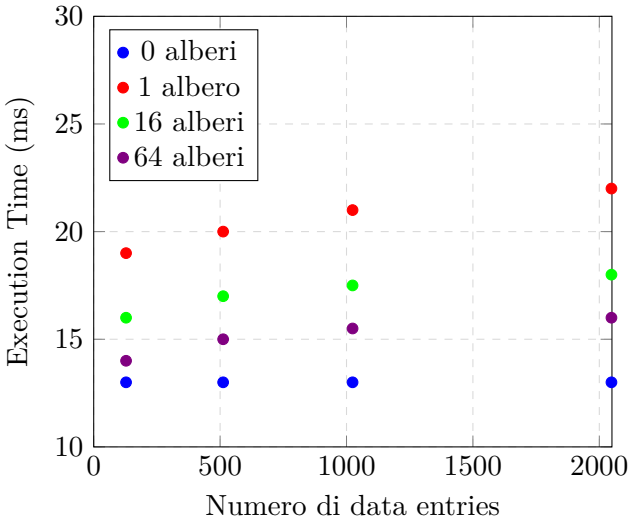

Figura 6.5: Grafico di confronto per Execution Time nella verifica

6.2 Generalizzazione della funzione costo e applicazioni

Ricordando le definizioni di N e M presentate nel paragrafo 6.1, è stata definita una funzione costo $C_{(N,M)}$ caratterizzata da 3 termini pesati dai coefficienti α, β e γ che verranno settati in modo da seguire le specifiche esigenze dell'azienda. I 3 fattori sono:

- Il costo delle transazioni in blockchain; ha andamento lineare in funzione di M.
- Il costo dello storage degli alberi, espresso in Megabyte, definito come $(2N - M) \times 32 \times 10^{-6}$.
- Il costo relativo alla fase di verifica di integrità, inteso come numero di hash da svolgere per ottenere la Merkle root attraverso la proof. Ne segue che, come la proof, questo contributo cresca con andamento logaritmico rispetto al numero di foglie di ogni albero.

Perciò, la funzione costo globale sarà caratterizzata dall'espressione (6.4)

$$C_{(N,M)} = \alpha \cdot M + \beta \cdot (2N - M) \cdot 0.000032 + \gamma \cdot (\log_2 N - \log_2 M) \qquad (6.4)$$

Nel caso in cui si considera la soluzione che non ricorre ai Merkle tree, l'unico termine rilevante risulta essere quello relativo al costo in blockchain, ma in questo caso la crescità lineare è rispetto a N. In tal caso la funzione costo assumerà la forma:

$$C_N = \alpha \cdot N \tag{6.5}$$

L'idea è quella di applicare tali funzioni ad alcuni casi di studio teorici al fine di individuare il valore di M con cui poter minimizzare i costi.
A tale scopo, per l'ottimizzazione numerica è stata utilizzata la funzione "minimize_scalar" offerta dalla libreria SciPy per minimizzare una funzione di una singola variabile. Sono poi stati individuati 3 diversi casi di studio in cui poter applicare tale procedura.

1. Azienda con numerosi data entry, database interno gratuito e budget limitato. β assumerà perciò un valore tendente allo zero in quanto non vi sono costi di storage; α sarà dominante rispetto a γ perché non sono stati espressi particolari vincoli sui tempi di verifica, mentre invece non si hanno ampie risorse economiche.
   Considerando i valori di $N = 32768$, $\alpha = 0.90009$, β =0.0000001 e $\gamma = 0.0999$, il numero ottimale di alberi risulta essere pari ad 1, in modo da limitare i costi in blockchain.

2. Azienda che deve pagare spazio cloud per database, ha pochi data entry e dispone di una blockchain privata.
   In questa casistica sarà il valore di α a tendere a zero poichè si utilizza una blockchain privata, mentre β assumerà un peso rilevante.
   Impostando $N = 32$, $\alpha = 0.0000001$, β =0.89999 e $\gamma = 0.1$, la soluzione ottima è non ricorrere a Merkle tree ma fare una transazione per ogni entry in modo da non appesantire il database con il salvataggio degli alberi.

3. Azienda con numerosi data entry, database interno gratuito, budget elevato e necessità di avere un tempo di verifica breve anche su dispostivi mobili con limitata potenza di calcolo.
   In quest'ultimo caso il coefficiente dominante sarà γ poichè il vincolo principale risiede nel tempo di verifica. .
   Configurando $N = 18432$, $\alpha = 0.00999$, β =0.0000001 e $\gamma = 0.9900$, l'ottimo si raggiunge costruendo 144 alberi.

Questi tre esempi applicativi mostrano come la scelta dell'architettura che minimizza la funzione costo dipende da molteplici fattori ed è unica per ogni casistica di studio.

Capitolo 7

Conclusioni

In conclusione, l'obiettivo centrale di questa tesi è stato quello quello di rendere possibile la comunicazione tra un database e la blockchain Ethereum allo scopo di certificare e tracciare dati ed individuare lo schema più vantaggioso per realizzare tale scopo.

Durante questa ricerca, sono state esaminate diverse architetture. Mediante simulazioni, sono stati analizzati e identificati i rispettivi punti di forza e di debolezza di ciascuna architettura.

Il vantaggio chiave di organizzare i dati in uno o più Merkle tree è di limitare in modo considerevole il numero di transazioni da inviare alla blockchain, riducendo così in modo significativo i costi di Ethereum a discapito di un aumento di storage nel database e di una fase di verifica di integrità che necessita di un numero maggiore di operazioni.

Cercando di generalizzare la trattazione si è poi svolta un'analisi attraverso l'introduzione di una funzione costo parametrica con la quale si è andati ad individuare la soluzione ottimale per diversi casi di studio teorici. Si è notato come, al variare della scelta dei fattori moltiplicativi che pesano i vari contributi, la scelta ottima varia e va a toccare tutte le diverse architetture presentate in questo lavoro di tesi.

Questo risultato è molto importante poiché evidenzia come la soluzione ottimale dello schema dipenda essenzialmente dalle singole esigenze dell'azienda e dalle proprie caratteristiche e non esiste perciò una configurazione ideale a priori.

Bibliografia

[1] Dlt. https://en.wikipedia.org/wiki/Distributed_ledger.

[2] Ledger. https://www.techtarget.com/searchcio/definition/ledger-database.

[3] Rose Kavitha. Advancement of bitcoin enabling smart governance: A case study of innovating blockchain technology. 56:2328–2333, 2022.

[4] Blockchain. https://en.wikipedia.org/wiki/Blockchain.

[5] La classificazione delle blockchain. https://www.spindox.it/it/la-classificazione-delle-blockchain/#:~:text=Esistono%20principalmente%20tre%20tipologie%20di,registri%20personalizzati%20per%20applicazioni%20specifiche.

[6] Bitcoin. https://academy.youngplatform.com/criptovalute/bitcoin-cos-e/.

[7] Aspetto storico di ethereum. https://brightnode.io/it/la-storia-e-la-crescita-del-progetto-ethereum/.

[8] Vitalik Buterin. Ethereum white paper. 2018.

[9] Proof-of-stake. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#:~:text=In%20Ethereum%27s%20proof%2Dof%2Dstake,and%20propagating%20new%20blocks%20themselves.

[10] Transazione ethereum. https://ethereum.org/it/developers/docs/transactions/.

[11] Smart contracts. https://ethereum.org/gl/developers/docs/smart-contracts/.

[12] Nft. https://thecryptogateway.it/nft/.

[13] A.F. Mendi. Blockchain for food tracking. *Electronics*, 11:2491, 2022.

[14] Food trust. https://www.ibm.com/blockchain/solutions/food-trust.

[15] hyperledge. https://hyperledger-fabric.readthedocs.io/it/latest/whatis.html#:~:text=La%20piattaforma%20Fabric%20ÃÍ%20anche,e%20quindi%20completamente%20non%20attendibili.

[16] R. Kamath. Food traceability on blockchain: Walmart's pork and mango pilots with ibm. *J. Br. Blockchain Assoc*, 1:1–12, 2018.

[17] Feng Tian. A supply chain traceability system for food safety based on haccp, blockchain and internet of things. pages 1–6, 2017.

[18] et al. McConaghy, T. Bigchaindb: A scalable blockchain database. bigchaindb-whitepaper. 2016.

[19] Miguel Pincheira Caro, Muhammad Salek Ali, Massimo Vecchio, and Raffaele Giaffreda. Blockchain-based traceability in agri-food supply chain management: A practical implementation. pages 1–4, 2018.

[20] Emmanuel Nyaletey, Reza M. Parizi, Qi Zhang, and Kim-Kwang Raymond Choo. Blockipfs - blockchain-enabled interplanetary file system for forensic and trusted data traceability. pages 18–25, 2019.

[21] Blockcerts. https://www.blockcerts.org/.

[22] Truerec. https://news.sap.com/2017/07/meet-truerec-by-sap-trusted-digital-credentials-powered-by-blockchain/.

[23] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. Certchain: Public and efficient certificate audit based on blockchain for tls connections. pages 2060–2068, 2018.

[24] Ethereum smart contracts for educational certificates. https://core.ac.uk/reader/211084230.

[25] Yifan Wu Rujia Li. Blockchain based academic certificate authentication system overview.

[26] Javascript. https://en.wikipedia.org/wiki/JavaScript.

[27] web3.js. https://web3js.readthedocs.io.

[28] crypto.js. https://www.npmjs.com/package/crypto-js.

[29] mongoose.js. https://mongoosejs.com.

[30] os-utils. https://www.npmjs.com/package/node-os-utils.

[31] web3. https://web3js.readthedocs.io/en/v1.10.0/index.html.

[32] Ganache. https://trufflesuite.com/docs/ganache/.

[33] Funzione hash. https://it.wikipedia.org/wiki/Funzione_crittografica_di_hash#Verifica_dell'integritÃ_di_un_messaggio.

[34] Funzione suriettiva. https://it.wikipedia.org/wiki/Funzione_suriettiva.

[35] Collisioni e proprietà funzione hash. https://it.wikipedia.org/wiki/Collisione_hash.

[36] Sha. https://en.wikipedia.org/wiki/Secure_Hash_Algorithms.

[37] Merkle-damgård. https://it.wikipedia.org/wiki/Costruzione_di_Merkle-DamgÃërd.

[38] Sponge construction. https://keccak.team/sponge_duplex.html.

[39] Andamento prezzo ether. https://www.google.com/finance/quote/ETH-EUR?sa=X&ved=2ahUKEwjwm7-_guSCAxW-X_EDHVx8CIwQ-fUHegQIDhAf.

[40] Merkle tree. https://en.wikipedia.org/wiki/Merkle_tree.

Ringraziamenti

Giunto alla conclusione di questo entusiasmante percorso, desidero esprimere la mia profonda gratitudine alle persone che mi hanno accompagnato in questo stimolante viaggio.

Un ringraziamento speciale al professor Marco Baldi, che ha saputo trasmettermi la sua passione verso i suoi insegnamenti, offrendomi un sostegno prezioso nella realizzazione di questa tesi. La sua dedizione ha dimostrato che dietro a un grande professore si cela sempre una grande persona.

Un caloroso ringraziamento va all'intero team che mi ha affiancato in questi mesi: Paolo Santini, Giulia Rafaiani, Massimo Battaglioni e Giacomo Zonneveld. Gran parte di questo lavoro è il risultato della vostra collaborazione e del vostro costante supporto. Siete stati un punto di riferimento per me, sempre pronti a risolvere ogni dubbio e incertezza.
Scusami Giulia per tutti gli accenti che ti ho fatto correggere, ora finalmente, dopo 24 anni di errori, ho imparato (spero) a distinguerli.

Ai miei genitori che mi hanno sempre incoraggiato nel perseguire i miei obiettivi, grazie dell'amore che mi mostrate quotidianamente, grazie per non avermi mai fatto mancare nulla, grazie per l'educazione che mi avete dato.
La persona che sono oggi è merito vostro, e spero di rendervi sempre fieri di me.

A mio fratello Luca, grazie per la tua infinita disponibilità che mi dimostri, grazie per sopportarmi tutti i giorni, immagino che sia un lavoro difficile ma qualcuno dovrà pur farlo.

A mia sorella Silvia, che non è stata una semplice sorella, ma soprattutto una seconda mamma. Con la tua forza e determinazione mi hai insegnato come superare i mille ostacoli che la vita ti pone inesorabilmente davanti. Sei sempre stata un'ispirazione per me e spero che io possa essere lo stesso per le tue figlie Ariel e Diana, una presenza luminosa che accompagna il cammino della loro crescita.

Ai miei nonni, ricordi indelebili del passato, che porterò sempre nel mio cuore.

A tutti i miei amici, compagni di mille avventure, di mille pazzie, grazie per la vostra presenza costante e per dimostrami giorno dopo giorno l'importanza di condividere esperienze con delle persone speciali come voi. Ci sono amicizie che diventano famiglia, e io ho avuto la fortuna di viverlo sulla mia pelle.

A chi c'è sempre stato nonostante tutto, a chi sempre ci sarà, a chi nel suo piccolo rende le mie giornate migliori, a chi decide di spendere il suo tempo standomi vicino, a chi mi dimostra amore.
A voi devo tutto e vi ringrazio di cuore, vi dedico questo speciale traguardo.

Ancona, dicembre 2023

Marco Farinasso