

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in INGEGNERIA INFORMATICA

Tesi di Laurea

Gestione degli asset digitali e astrazione delle operazioni di supply chain tramite API REST ed ERC1155

Relatore: Chiar.mo Prof. Melchiori Michele

Correlatori: Chiar.mo Prof. Bianchini Devis

Laureando:
Sica Mattia

Anno Accademico 2024/2025

Ringraziamenti

Un sentito ringraziamento ai Professori Michele Melchiori e Devis Bianchini, per avermi guidato con competenza e dedizione nell'esplorazione di un tema tanto stimolante quanto formativo.

Il mio pensiero più affettuoso va ai miei nonni, Antonio e Giuseppina, a mio padre Roberto, a mia madre Isis, e alle mie zie Adelia e Maria. Il loro sostegno incondizionato ha rappresentato un punto fermo nel mio percorso, accademico e umano. Anche quando la strada si è fatta più impervia, la loro vicinanza non è mai venuta meno, e di questo sarò loro sempre grato.

Ringrazio gli amici che quotidianamente hanno condiviso con me gioie e fatiche, e quelli che, pur nella distanza, restano presenze vive e care.

A chi ha creduto nel programma Erasmus rivolgo un ringraziamento profondo: questa esperienza mi ha arricchito non solo professionalmente, ma soprattutto come persona, lasciandomi un patrimonio di esperienze, amicizie e ricordi che custodirò per sempre.

Infine, grazie a Luiza, che da quando ho messo piede ad Almería mi accompagna ogni giorno, anche con un oceano di distanza.

Introduzione

Contesto e motivazioni
Obiettivi del lavoro
Struttura della tesi

1. Supply chain: definizione e requisiti

1.1. Definizione di supply chain e caratteristiche
1.2. Digital Product Passport

2. Modello di tracciabilità dei lotti

2.1. Panoramica sul modello Cerere
2.2. Rappresentazione di un lotto
2.3. Rappresentazione delle attività
2.4. Astrazione delle attività
2.5. Ciclo di vita di un lotto

3. Tokenizzazione

3.1. Introduzione ai token e standard
3.2. Open Zeppelin e Standard ERC
3.3. Standard ERC1155 per la rappresentazione dei lotti
3.4. Transazioni ed Eventi

4. Analisi dei requisiti: modello dei dati

4.1. Struttura di un lotto
4.2. Mappatura tra operazioni astratte e operazioni sui token
4.3. Definizione del modello dei dati

5. Design dell'architettura del sistema

5.1. Tecnologie e strumenti adottati
5.2. API RESTful come interfaccia di interoperabilità
5.3. Architettura core del sistema
5.4. Il modello a tre livelli di storage: blockchain, IPFS e MongoDB
5.5. Smart contract

6. Implementazione

- **6.1.** Endpoint esposti: struttura generale ed esempi
- **6.2.** Implementazione degli endpoint e diagrammi UML di sequenza
- **6.3.** Endpoint addizionali

7. Validazione e benchmark

- **7.1.** Definizione e analisi dei test case
- **7.2.** Benchmarking
- **7.3.** Considerazioni sulle Prestazioni
- **7.4.** Considerazioni sui Costi

8. Estensioni, Criticità e sviluppi futuri

- **8.1.** Soluzioni Layer 2
- **8.2.** Gestione delle chiavi utente
- **8.3.** Generazione degli ID
- **8.4.** Multisignature e non-repudiation

Conclusioni

Sintesi dei risultati
Limiti del modello
Prospettive di ricerca

Figura 1: Traceability Package del modello Cerere 8
Figura 2: Attività di cessione/acquisizione 9
Figura 3: Rappresentazione dei sei schemi di attività nel modello Cerere ... 10
Figura 4: Macchina a stati del lotto nel modello Cerere 11
Figura 5: Diagramma di attività UML del lotto 12
Figura 6: Tabella comparativa degli standard ERC 16
Figura 7: Principali parametri di una transazione ETH 19
Figura 8: Flusso di una transazione ETH 20
Figura 9: Mappatura operazioni 25
Figura 10: Modello dei dati 26
Figura 11: UI Ganache 29
Figura 12: UI Pinata 30
Figura 13: Alcuni esempi di URI esposti 31
Figura 14: Architettura del Sistema 32
Figura 15: Architettura a tre livelli di Storage 34
Figura 16: Visione Semplificata dello Smart Contract 36
Figura 17: Diagramma di Stato di Contaminazione 37
Figura 18: Alcuni Endpoint per la gestione di lotti 40
Figura 19: Alcuni Endpoint per la gestione di un lotto 40
Figura 20: Alcuni codici errore restituiti dall'API 41
Figura 21: Alcuni codici errore restituiti dall'API (II) 41
Figura 22: Creazione di un batch 42
Figura 23: Diagramma di sequenza per la creazione 44
Figura 24: Trasferimento di un batch 45
Figura 25: Estratto .env, credenziali simulate su Ganache 45
Figura 26: Diagramma di sequenza per il trasferimento 46
Figura 27: Split di un batch 47
Figura 28: Diagramma di sequenza per lo split 48
Figura 29:Merge di un batch 49
Figura 30: Diagramma di sequenza per il Merge 50
Figura 31: Transform di un batch 51
Figura 32: Diagramma di sequenza per il Transform 52
Figura 33: Burn di un batch 55
Figura 34: Diagramma di sequenza per il Burn 55
Figura 35: Validate di batches 57
Figura 36: Diagramma di sequenza di Validate 58
Figura 37: Incidente di contaminazione 59
Figura 38: Contaminate di un batch 60
Figura 39: Esempio di Contaminazione 61
Figura 40: Diagramma di sequenza di Contaminate 62
Figura 41: Endpoint con valori aggregati 66
Figura 42: Tempo totale medio per Endpoint 68

Figura 43: Breakdown dei tempi per Endpoint 68
Figura 44: Comparazione costi operazioni..... 69
Figura 45: Tx.await(i) confronto Ethereum / Polygon 72
Figura 46: Confronto costi operazioni tra Ethereum e Polygon 72
Figura 47: Confronto modelli Custodial e Self-custody..... 73
Figura 48: Confronto tra possibili soluzioni per la generazione degli ID 75

Introduzione

La supply chain rappresenta un ambito intrinsecamente complesso e multidimensionale, in cui si intrecciano vincoli di varia natura. La sua ottimizzazione è al medesimo tempo una sfida e un'opportunità strategica in quanto regola aspetti come efficienza e affidabilità dei sistemi produttivi e distributivi contemporanei. Questa tesi si propone di sviluppare una metodologia per la modellizzazione della supply chain come infrastruttura **phygital**: introdurre un modello di astrazione delle operazioni e delle componenti fondamentali come possono essere i lotti, traslarli mediante una mappatura su ambiente tokenizzato, progettare e sviluppare un'architettura per concretizzare l'approccio e, infine, validarlo mediante sperimentazioni e analisi.

Contesto e motivazioni

Il lavoro svolto si colloca nel quadro della progressiva digitalizzazione promossa a livello europeo, dove la gestione di ambiti centrali per la società contemporanea quali l'identità digitale, la gestione della valuta e la tracciabilità dei prodotti, rappresenta una priorità politica ed economica. In tale contesto, le tecnologie Distributed Ledger (DLT) sono oggetto di crescente attenzione istituzionale, come dimostrano le iniziative quali l'euro digitale e i progetti di identità elettronica. Nei capitoli a seguire, si propone un approccio metodologico per la modellizzazione di architetture di supply chain token based, basato sull'integrazione tra gli standard blockchain e le interfacce REST, con particolare riferimento allo standard ERC-1155 di Ethereum, che viene analizzato nelle sue caratteristiche e nelle conseguenti potenziali applicazioni alla gestione delle catene di approvvigionamento. La metodologia si sviluppa a partire dall'analisi dei requisiti dei sistemi di supply chain, prosegue con una valutazione comparativa tra le funzionalità dei token ERC-1155 rispetto a quelle di altri standard e giunge alla mappatura delle operazioni di supply chain sulle primitive ERC-1155, con l'obiettivo di individuare configurazioni architetturali adeguate. La validazione di quanto sviluppato è stata condotta mediante una fase sperimentale che ha incluso la misurazione delle prestazioni e dei costi operativi del sistema implementato, con benchmark sulle tempistiche di esecuzione, analisi dell'efficienza dei servizi esterni integrati nell'API e confronti con soluzioni basate su blockchain alternative, in particolare layer 2 come Polygon, in modo da fornire una base empirica su cui elaborare delle considerazioni in merito alla fattibilità e all'efficacia dell'approccio nei contesti applicativi considerati.

Obiettivi di lavoro

Gli obiettivi principali della ricerca possono essere sintetizzati come segue:

- **Identificazione dei costrutti fondamentali della supply chain**, finalizzata all'identificazione e alla modellizzazione esaustiva delle entità e delle operazioni minime che caratterizzano una catena di approvvigionamento.
- **Mappatura delle entità e delle operazioni della supply chain in ambiente tokenizzato**, in particolare rispetto alle primitive offerte dallo standard ERC-1155.

- **Progettazione e sviluppo di un modello dati e di una API REST** per l'interazione con un sistema di gestione della supply chain token-based, integrando funzionalità on-chain con strumenti di gestione off-chain.
- **Validazione sperimentale** dei risultati in termini di tempi, costi e scalabilità, attraverso testing e benchmarking degli endpoint esposti.
- **Individuazione di limiti, criticità e potenzialità** del modello proposto, con prospettive di estensione e applicabilità a scenari futuri.

Struttura della tesi

La tesi è articolata in otto capitoli principali:

- **Capitolo 1**: analisi del concetto di supply chain, del contesto normativo europeo e delle direttrici di trasformazione digitale, con particolare riferimento alla legittimazione delle tecnologie DLT per la tutela di interessi strategici.
- **Capitolo 2**: introduzione del modello astratto denominato ***Cerere***, utilizzato come riferimento per l'astrazione delle operazioni di business e delle entità fondamentali coinvolte nelle catene di approvvigionamento.
- **Capitolo 3**: presentazione degli standard ERC di Ethereum, delle librerie OpenZeppelin e delle dinamiche di funzionamento delle transazioni di token, così da giustificare le modalità di traduzione della business logic astratta in aggregazioni di primitive token based.
- **Capitolo 4**: descrizione della mappatura tra le operazioni astratte di supply chain e le operazioni *on-chain*; definizione degli endpoint principali e del modello dei dati adottato.
- **Capitolo 5**: illustrazione dell'architettura complessiva del sistema, dei servizi impiegati (API REST, IPFS, MongoDB, blockchain), nonché delle principali funzioni implementate nello smart contract.
- **Capitolo 6**: presentazione dei diagrammi UML di sequenza relativi agli endpoint e descrizione del contenuto e del flusso delle richieste.
- **Capitolo 7**: analisi sperimentale mediante suite di test, unitari ed end-to-end, e benchmarking. Comparazione con le prestazioni attese su blockchain alternative in termini di costi e alle latenze.
- **Capitolo 8**: discussione dei limiti della soluzione proposta, delle criticità riscontrate e delle potenzialità di sviluppo futuro.
- **Conclusioni**: vengono riassunti i risultati ottenuti, evidenziando i contributi scientifici e pratici della ricerca e tracciando le possibili direzioni di lavoro futuro.

Capitolo 1: Supply chain, definizione e requisiti

La supply chain costituisce uno degli ambiti più complessi della gestione industriale poiché riflette l'interdipendenza dei sistemi produttivi e distributivi globali: da modello lineare orientato esclusivamente all'efficienza operativa, questa si è progressivamente trasformata in un complesso ecosistema in cui flussi materiali e informativi devono essere coordinati in modo integrato [1][2]. Trasparenza e tracciabilità rappresentano in tale contesto un requisito **imprescindibile** sia per aspetti quali la conformità ai nuovi indirizzi normativi europei, sia per la competitività delle imprese. In questo contesto, l'introduzione del *Digital Product Passport* da parte della Commissione Europea segna un passaggio fondamentale: istituisce l'obbligo di un'identità digitale per i prodotti, principio che abilita forme avanzate di gestione della supply chain e che, indirettamente, identifica le Distributed Ledger Technologies (DLT) come tecnologie ideali per concretizzare una visione trasparente, interoperabile ed affidabile della supply chain.

Definizione di supply chain e caratteristiche

Il concetto di catena di approvvigionamento è evoluto in modo significativo nel tempo: le prime formalizzazioni teoriche definivano le supply chain come un'integrazione sequenziale di processi chiave, dal fornitore al cliente finale, enfatizzando l'ottimizzazione dei costi e l'efficienza operativa [1] ma questa visione lineare, si è progressivamente ampliata in un'ottica in cui si concepisce come sistema integrato orientato alla creazione di valore attraverso il coordinamento di flussi materiali e immateriali [2] [3]. In questa visione, la gestione delle relazioni tra attori si configura come l'elemento critico per la performance sistemica, **imponendo l'adozione di architetture organizzative robuste e di protocolli di coordinamento capaci di sostenere l'interoperabilità e la cooperazione necessaria tra soggetti eterogenei.** La natura poliedrica della supply chain rende quindi indispensabile per la sua modellizzazione digitale la definizione di astrazioni teoriche coerenti che consentano di concepire la catena come aggregazione di costrutti e operazioni standard, idonei a supportare implementazioni scalabili sia per la tracciabilità che per la gestione efficiente. All'interno di questa traiettoria evolutiva emergono quattro esigenze imprescindibili che rappresentano gli attuali cardini della gestione delle supply chain:

- **Trasparenza**: si configura come la capacità di ***rendere accessibili e affidabili le informazioni lungo l'intero ciclo di vita del prodotto***, requisito essenziale non soltanto per garantire conformità normativa, ma anche per consolidare la fiducia degli attori e abilitare forme di responsabilità condivisa.

- **Tracciabilità**: intesa come ***capacità di ricostruire o seguire la distinta base di lotti/prodotti sia a ritroso verso le origini sia in avanti lungo la distribuzione****,* si presenta come dimensione trasversale che abilita tutte le altre funzionalità, consentendo il monitoraggio integrale della catena, la prevenzione dei rischi e l'attuazione di eventuali richiami selettivi.

- **Resilienza**: definita come **capacità di mantenere livelli adeguati di servizio anche in presenza di situazioni critiche**, requisito strutturale per catene globali fortemente interdipendenti e vulnerabili.

- **Sostenibilità**: **riduzione degli impatti ambientali e sociali delle attività produttive e logistiche**, principio la cui rilevanza è in continua crescita, in linea con le più recenti direttive europee e con l'introduzione di strumenti quali il Digital Product Passport [4].

In questa prospettiva, la tracciabilità diventa il principio abilitante attraverso cui questi valori eterogenei trovano unificazione e continuità: la supply chain evolve verso un modello in cui i flussi si integrano in strutture condivise, capaci di garantire verificabilità e responsabilità diffusa lungo l'intero ciclo di vita del prodotto.

Digital Product Passport

Il **Digital Product Passport (DPP)**, introdotto dal *Regolamento (UE) 2024/1781*[1] quale identità digitale obbligatoria e persistente per ciascun prodotto, rappresenta l'attuale vertice evolutivo di un percorso di istituzionalizzazione della tracciabilità che trova le sue radici nel primo regolamento organico in materia, il *Regolamento (CE) n. 178/2002*[2]. Tale regolamento *(178/2002)*, adottato quale risposta sistemica a crisi alimentari[3] che avevano evidenziato le vulnerabilità delle catene di approvvigionamento europee introduce per la prima volta l'obbligo, nel territorio europeo, di garantire la tracciabilità dei prodotti alimentari attraverso tutte le fasi della produzione, trasformazione e distribuzione, sancendo l'obbligo per ciascun attore di identificare la provenienza e la destinazione dei beni immessi o ricevuti, introducendo un paradigma di responsabilità condivisa e di controllo informativo come funzionalità nativa delle filiere europee. L'impianto normativo, originariamente circoscritto al comparto agroalimentare, costituisce il nucleo fondante di una visione sistemica in cui la supply chain viene concettualizzata come sistema dinamico eterogeneo, caratterizzato da entità autonome ma interconnesse, la cui affidabilità richiede architetture informative interoperabili capaci di ridurre asimmetrie informative.

L'evoluzione normativa ha progressivamente ampliato l'ambito di applicazione di questi principi e rafforzato i meccanismi di controllo, culminando nel più recente e comprensivo Regolamento (UE) 2024/1781, denominato **ESPR *Ecodesign for Sustainable Products Regulation***, che rappresenta un incremento qualitativo di scala e ambizione, collocandosi nell'ambito della ***"Digital Decade"*** *delineata dalla Commissione europea* [5] [6] e introducendo quale strumento abilitante il ***Digital Product Passport (DPP)***, definito come ***"identità digitale obbligatoria, interoperabile e persistente di ciascun prodotto"***. L'ESPR e il DPP si inseriscono a loro volta nell'ampio piano di digitalizzazione europeo che tra le varie iniziative comprende *l'**European Digital Identity Wallet*** [7]*, concepito per fornire a cittadini e imprese un'identità digitale unitaria e sicura valida in tutta l'Unione, e* l'***Euro digitale***, *per il quale è stata presentata nel 2023 una proposta normativa che **riconosce esplicitamente le Distributed Ledger Technologies (DLT) come infrastrutture tecnologiche abilitanti*** [8]. **Le DLT emergono come soluzione trasversale e unificante per l'intera architettura della Digital Decade**: nell'ambito dell'euro digitale, garantiscono l'immutabilità e verificabilità delle transazioni; per l'European Digital Identity Wallet, assicurano autenticità e portabilità delle identità digitali; nel contesto del DPP, abilitano la creazione di registri distribuiti e immutabili per la tracciabilità dei prodotti, configurandosi come infrastruttura tecnologica fondamentale per la sovranità digitale europea. Il DPP, quale componente di questa architettura integrata e destinato a copertura progressiva di tutte le categorie merceologiche, si fonda su requisiti minimi comuni che ne determinano la natura sistemica: identificativo univoco persistente del prodotto e degli operatori; collegamento a data carrier fisico machine-readable conforme a standard ISO/IEC;

[1] *noto anche come **ESPR (Ecodesign for Sustainable Products Regulation)**, è una normativa europea che stabilisce un quadro per l'**ecodesign** dei prodotti.*

[2] *normativa europea fondamentale che stabilisce i **principi generali e requisiti della legislazione alimentare.***

[3] *le crisi della BSE ("mucca pazza") degli anni '90 e della diossina (Belgio, 1999) misero in evidenza la mancanza di sistemi efficaci di rintracciabilità lungo le filiere alimentari e furono tra i principali fattori che portarono all'adozione del Regolamento (CE) 178/2002.*

inclusione di contenuti informativi standardizzati comprendenti codici GTIN e TARIC[4], dichiarazioni di conformità, parametri ambientali, sostanze rilevanti e specifiche per riparazione, riuso e riciclo; interoperabilità tecnica mediante formati aperti privi di vendor lock-in; gestione multilivello degli accessi con visibilità differenziata per consumatori, operatori, autorità e dogane; resilienza sistemica assicurata da backup presso fornitori indipendenti e registro centrale europeo degli identificativi.

L'adozione di tutte queste iniziative nel loro insieme costituisce, di fatto, l'embrione di un'infrastruttura europea che mira all'integrazione di identità digitali di cittadini, imprese, prodotti, processi e transazioni economiche in un'architettura quanto più possibile uniforme, interoperabile, tracciabile e coerente, ridefinendo la supply chain come ecosistema informativo in cui tra le varie implicazioni, i prodotti corrispondono a delle entità digitali e il cui ciclo di vita materiale si riflette nel ciclo di vita di un digital twin interrogabile e verificabile [6][7]. In questa prospettiva sistemica, l'implementazione del DPP assume valenza che trascende l'ecodesign, configurandosi come componente di una strategia di autonomia tecnologica, in cui ***i requisiti di integrità, autenticità e immutabilità dei dati trovano traduzione tecnica naturale in architetture distribuite.*** Le blockchain e le *Distributed Ledger Technology (DLT)*[5] manifestano infatti rilevanza trasversale in quanto abilitano la rappresentazione di entità sotto forma di token crittografici corredati di metadati, costituendo la soluzione naturale per l'assolvimento degli obblighi regolatori.

L'adozione di tecnologie digitali per la tracciabilità richiede una preliminare fase di formalizzazione concettuale: l'eterogeneità delle catene di fornitura e la complessità dei flussi impongono infatti la ***definizione di modelli astratti capaci di ridurre la complessità sistemica a componenti trattabili***, dotati di semantica condivisa e valore esplicativo. In questo quadro si inserisce il modello *Cerere*, elaborato come schema concettuale per la tracciabilità della filiera alimentare ma caratterizzato da una generalità tale da consentirne l'estensione a diversi domini industriali [9].

[4] *GTIN: codice numerico standardizzato (GS1) che identifica in modo univoco prodotti e beni a livello globale. TARIC: banca dati UE che integra dazi, misure tariffarie e normative applicabili alle merci importate.*

[5] *Distributed Ledger Technology, indica un'infrastruttura digitale in cui i dati vengono registrati, condivisi e sincronizzati in modo decentralizzato tra più nodi di una rete.*

Capitolo 2: Modello di tracciabilità dei lotti

La definizione di un modello astratto rappresenta un passaggio fondamentale per procedere con la formalizzazione della logica operativa delle supply chain così da consentirne la traduzione in architetture digitali. In tale prospettiva, il modello **Cerere**, sviluppato da Cimino et al. (2005), costituisce un riferimento consolidato per la descrizione di entità e processi di filiera mediante un insieme essenziale di costrutti concettuali. La sua capacità di rappresentare in maniera unitaria lotti, attività e attori lo rende ideale come base concettuale per l'analisi e la modellizzazione di sistemi eterogenei. Qui, Cerere è adottato come schema di astrazione per la business logic della supply chain, astrazione sulla quale viene poi costruita la proposta di un sistema tokenizzato basato su blockchain.

Panoramica sul modello Cerere

Il modello *Cerere* rappresenta la formalizzazione di un approccio organico alla tracciabilità delle supply chain nel contesto della normativa europea introdotta dal Regolamento (CE) n. 178/2002[6]. Cerere si propone di descrivere in maniera unitaria i processi di produzione, trasformazione e distribuzione attraverso un insieme ridotto ma fondamentale di costrutti concettuali; *l'elemento distintivo del modello è la capacità di astrarre la complessità delle interazioni di filiera mediante strumenti formali come diagrammi UML e documenti XML interoperabili, con l'obiettivo di garantire la trasparenza informativa lungo l'intera catena e la responsabilità condivisa tra gli attori.* Concepito originariamente per il settore alimentare, mantiene un carattere generale che lo rende idoneo ad applicazioni in contesti industriali differenti e ne preserva la rilevanza metodologica anche a distanza di tempo. **L'interesse per Cerere risiede nella sua funzione di astrazione di base**: *i concetti cardine di lotto, attività e attore responsabile costituiscono la struttura elementare sulla quale è possibile costruire modelli teorici e implementazioni pratiche.* **L'analisi si concentra sugli aspetti di modellazione dei dati connessi alle entità principali del modello, vale a dire** il **lotto** e l'**attività**, con l'obiettivo di **evidenziare le caratteristiche informative imprescindibili** che ne consentono l'utilizzo come unità fondamentali di tracciabilità.

[6] *normativa europea fondamentale che stabilisce i* ***principi generali e requisiti della legislazione alimentare.***

Rappresentazione di un Lotto

La struttura concettuale del modello viene formalizzata in concreto con il *Traceability Package*, che rappresenta il nucleo informativo su cui si fonda la capacità del sistema di descrivere la storia dei prodotti lungo la supply chain. L'immagine mostra come il modello introduca l'astrazione di *entità tracciabile*, una classe generale dalla quale derivano due tipologie fondamentali: il *lotto* e l'*attività*. Questa scelta evidenzia che ***la tracciabilità, nel quadro di Cerere, non riguarda soltanto i beni materiali che attraversano la catena, ma anche i processi che ne determinano la trasformazione e la movimentazione***.

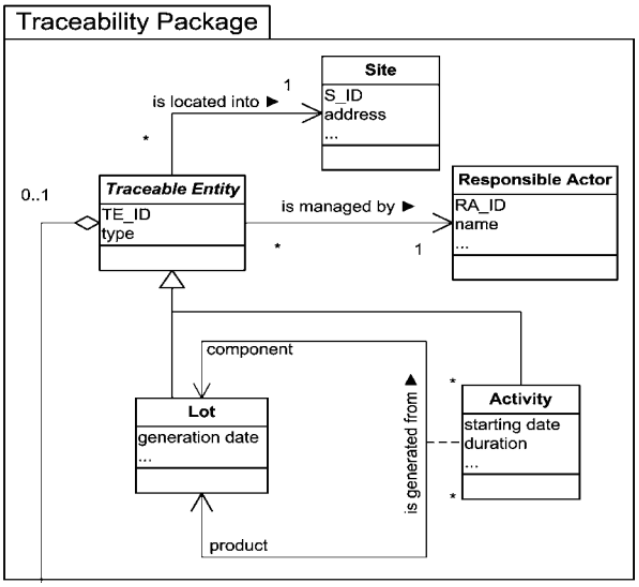


Figura 1: Traceability Package del modello Cerere

Il lotto rappresenta l'unità di prodotto e costituisce **l'oggetto primario della tracciabilità**. **L'attività** rappresenta invece **l'operazione che agisce su uno o più lotti**, con attributi come data di inizio e durata, e permette di **modellare i processi di trasformazione e movimentazione**. Il legame "is generated from" esplicita come **un lotto in uscita possa derivare da uno o più lotti in ingresso attraverso l'esecuzione di un'attività**, consentendo la ricostruzione a ritroso della catena di eventi. Elemento cardine della modellazione è l'**identificatore univoco (TE_ID)**, che assicura la possibilità di distinguere senza ambiguità ogni entità tracciabile lungo l'intera filiera. Ogni entità deve essere associata a un **attore responsabile (Responsible Actor)**, individuato da un **identificativo proprio (RA_ID)** e caratterizzato da attributi quali il nome e il ruolo nella filiera. Tale legame garantisce che la responsabilità informativa e operativa di ciascun lotto o attività sia sempre attribuibile a un soggetto definito. Parallelamente, ciascuna entità deve essere associata a un **sito (Site)**, identificato da un codice **(S_ID)** e descritto tramite attributi come l'indirizzo o la funzione logistica. Il modello permette quindi di descrivere la **rete di relazioni** che lega prodotti, operazioni, attori e siti, rendendo possibile non solo identificare un lotto come entità isolata, ma inserirlo in un sistema dinamico di responsabilità e trasformazioni.

Rappresentazione delle attività

Le **attività** rappresentano **le operazioni fondamentali** che determinano le azioni sui lotti lungo la supply chain: costituiscono l'anello di congiunzione tra input e output e consentono di descrivere in maniera unitaria **sia le operazioni di natura fisica** (trasporto, trasformazione, assemblaggio) **sia quelle di natura transazionale** (acquisizione, vendita, distribuzione). **Ogni attività è concepita come entità tracciabile dotata di un identificatore univoco**, capace di collegare i lotti coinvolti con gli attori responsabili e i siti in cui l'operazione si svolge.

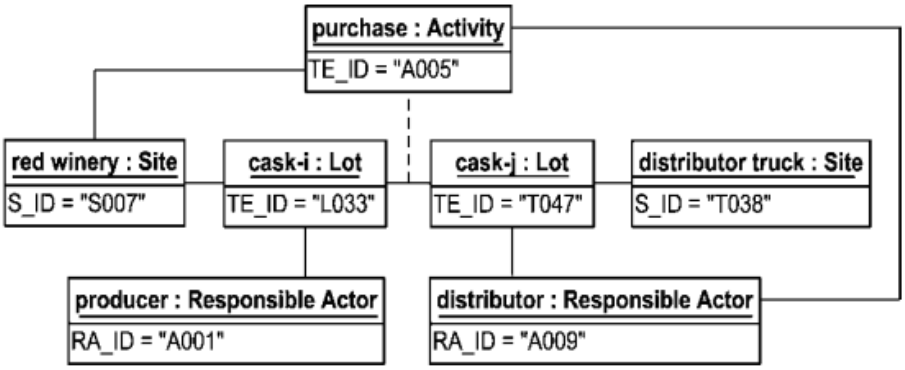


Figura 2: Attività di cessione/acquisizione

L'esempio riportato illustra un'attività di **cessione/acquisizione**, in cui un produttore trasferisce a un distributore una botte di vino trasportata presso il magazzino del destinatario. L'attività è identificata come *purchase* (TE_ID = A005) e mette in relazione due lotti, *cask-i* e *cask-j*, che pur rappresentando il medesimo bene fisico sono trattati come entità distinte poiché ciascun attore responsabile assegna un proprio identificatore univoco. Oltre ai lotti, l'attività collega i due attori coinvolti (*producer* e *distributor*) e i rispettivi siti (*red winery* e *distributor truck*), configurandosi come nodo informativo che registra simultaneamente il flusso materiale e quello di responsabilità.

Astrazione delle attività

Cerere non si limita a rappresentare la struttura statica delle entità fondamentali della tracciabilità, ma introduce anche una formalizzazione del comportamento dinamico dei lotti, descrivendo le possibili trasformazioni a cui essi sono sottoposti lungo la filiera. Tale dinamica è espressa attraverso un insieme di schemi elementari, o ***activity patterns***, che catturano i modi in cui i lotti possono essere acquisiti, trasformati, suddivisi, integrati, alterati o movimentati. **Questi pattern costituiscono il nucleo concettuale che consente di generalizzare ogni operazione di supply chain in termini di relazioni fra lotti preesistenti e nuovi lotti generati**, mantenendo così la possibilità di eseguire sia processi di *tracing* a ritroso sia di *tracking* in avanti.

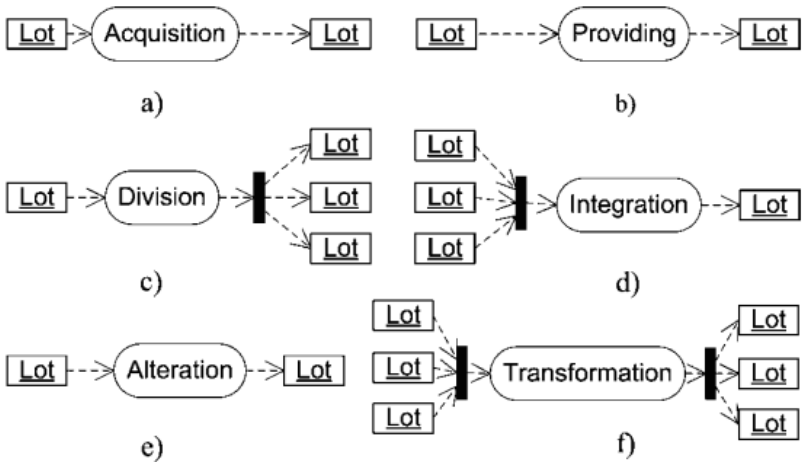


Figura 3: Rappresentazione dei sei schemi di attività nel modello Cerere

Nello specifico, vengono individuate un insieme di attività fondamentali che consentono di rappresentare in forma astratta le principali operazioni di filiera: **l'acquisizione** si verifica quando un attore riceve un lotto da un altro, generando un nuovo lotto collegato al preesistente e consentendo così di risalire alle origini del prodotto, lo schema complementare è la **cessione**, in cui un attore fornisce un lotto a un altro, generando un nuovo lotto che permette di seguirne il percorso lungo la catena. La **divisione** descrive il caso in cui un lotto viene suddiviso in più lotti, mentre **l'integrazione** formalizza la situazione opposta, ovvero la fusione di più lotti in un lotto unico, tipica delle attività di miscelazione o confezionamento. **L'alterazione** indica la generazione di un nuovo lotto attraverso una modifica delle caratteristiche del lotto originario. La **trasformazione** rappresenta invece il processo in cui più lotti in ingresso vengono sottoposti a una lavorazione che produce lotti in uscita. La generalizzazione proposta da *Cerere* consente quindi di modellizzare la varietà delle operazioni di filiera riducendole a un insieme limitato e coerente di schemi formali.

Ciclo di vita di un Lotto

Il modello astratto introduce una rappresentazione formale dell'evoluzione temporale dei lotti attraverso una macchina a stati finiti che cattura la dinamica operativa della supply chain. L'approccio adottato riconosce che ogni lotto, nel corso del suo percorso lungo la filiera, attraversa delle fasi distinte caratterizzate da specifiche condizioni di responsabilità. La formalizzazione proposta dal modello astratto identifica tre stati fondamentali: acquired, transformed e provided, che rappresentano le configurazioni essenziali in cui un lotto può trovarsi durante il suo ciclo di vita. La transizione tra questi stati non è arbitraria, ma segue delle regole che riflettono la logica operativa delle supply chain reali. Come illustrato nella *Fig. 4*, il modello stabilisce che un lotto acquisito può essere sottoposto a trasformazione, diventando transformed, oppure essere direttamente fornito ad altro attore, assumendo lo stato provided. Un lotto trasformato può essere fornito, mentre un lotto fornito può essere acquisito da un nuovo attore responsabile, ricominciando così il ciclo sotto una diversa responsabilità. Particolare significato assume il vincolo che impedisce il passaggio diretto da transformed ad acquired: questa restrizione deriva dalla scelta di trattare cessione e acquisizione come due attività distinte piuttosto che come un'unica transazione. In questa logica, un lotto trasformato deve necessariamente essere prima fornito dall'attore che detiene la responsabilità, e successivamente acquisito dal nuovo attore destinatario. Tale meccanismo, sebbene possa apparire ridondante, garantisce l'esplicitazione completa dei passaggi di responsabilità e preserva l'integrità della catena di custodia attraverso la formalizzazione di ogni trasferimento.

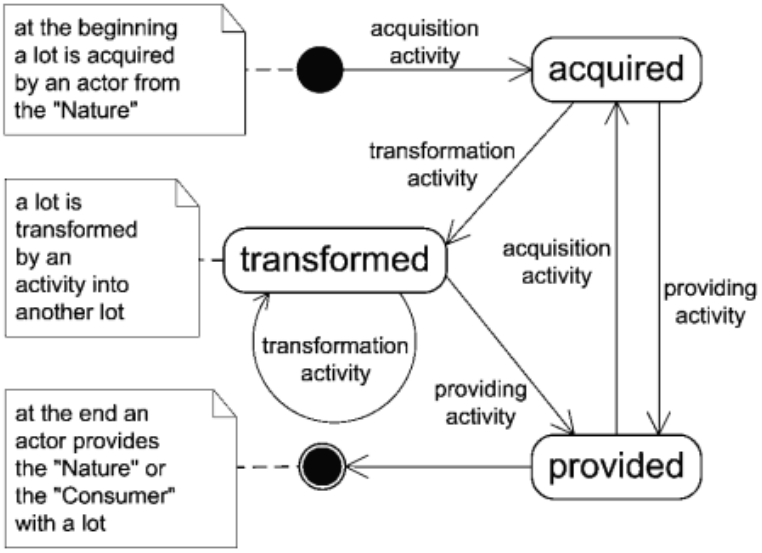


Figura 4: Macchina a stati del lotto nel modello Cerere

La rappresentazione mediante diagramma di attività UML *(Fig. 5)* arricchisce ulteriormente il modello introducendo tre domini funzionali: *Nature*, che rappresenta l'ambiente da cui provengono le materie prime e verso cui convergono i prodotti di scarto; *Responsible Actor*, che identifica il soggetto che detiene la responsabilità operativa del lotto; e *Consumer*, che rappresenta il punto finale del percorso del lotto specifico. Questa strutturazione evidenzia come ogni transizione di stato implichi non solo un cambiamento nelle caratteristiche del lotto in sé, ma soprattutto un trasferimento di responsabilità tra diversi attori.

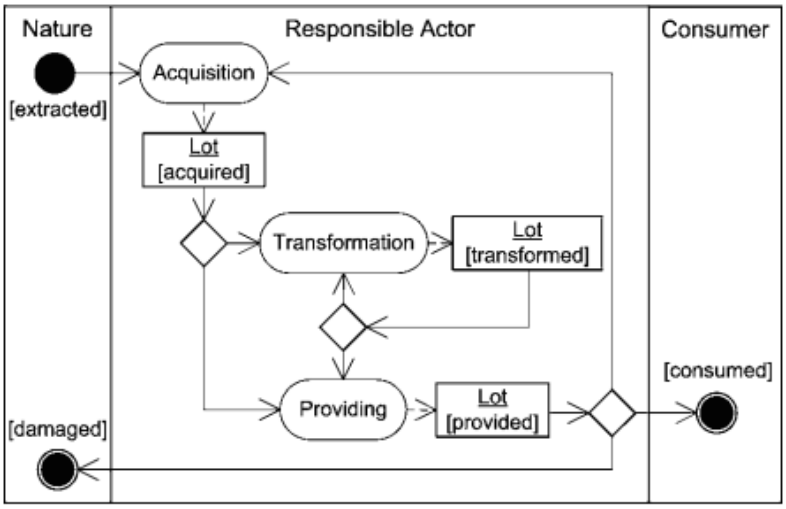


Figura 5: Diagramma di attività UML del lotto

Il modello astratto fornisce una solida base concettuale dalla quale estrarre i principi fondamentali per la progettazione di sistemi di tracciabilità digitali. Dal modello emergono tre elementi cardine che devono essere preservati in qualsiasi implementazione: i **lotti** come unità di prodotto chiaramente identificabili, gli **attori** come soggetti responsabili univocamente definiti, e le **operazioni** come attività che determinano le trasformazioni dei lotti e i passaggi di responsabilità tra attori lungo la filiera. La tassonomia delle operazioni di supply chain, cioè le operazioni di acquisizione, trasformazione, divisione, integrazione, alterazione e cessione risulta particolarmente significativa e costituisce il riferimento che adotteremo per il sistema token-based. Una volta illustrate le caratteristiche e le funzionalità dei token digitali nel capitolo successivo, sarà possibile infatti mappare ciascuna di queste operazioni sui meccanismi di gestione e trasferimento di token. L'alternanza tra gli stati **ceduto**, **acquisito** e **trasformato** rappresenta un altro concetto fondamentale che verrà riproposto nell'architettura tokenizzata, seppur con modalità differenti. I token digitali introducono infatti nuove possibilità operative che superano alcune rigidità del modello originario come l'unificazione di cessione e acquisizione in operazioni atomiche di trasferimento e la capacità di gestire lotti anche in forma parziale attraverso meccanismi di frazionamento. Questo significa che lo stato di un lotto non è più assoluto, ma può dipendere dalla sua suddivisione e dalla distribuzione delle sue parti tra diversi attori. L'analisi presentata nei prossimi capitoli illustrerà come questi strumenti digitali consentano di identificare e rappresentare i tre costrutti fondamentali e come i token possano essere utilizzati per riprodurre efficacemente i cicli di vita dei lotti all'interno di architetture blockchain distribuite.

Capitolo 3: Tokenizzazione

La formalizzazione concettuale offerta dal modello Cerere necessita di un'infrastruttura capace di tradurre le entità e le operazioni della supply chain astratta in primitive computabili e verificabili. In tale prospettiva, si analizzano i principali **standard di tokenizzazione Ethereum**. Questo capitolo introduce le principali famiglie di standard ERC, ne analizza le caratteristiche semantiche e operative e discute il ruolo di ERC-1155 come soluzione particolarmente adatta a rappresentare la business logic delle supply chain, ponendo le basi per la successiva trasposizione del modello astratto in architettura token-based.

Introduzione ai Token e Standard

In Ethereum, un token è ***una rappresentazione digitale programmabile di una risorsa o di un diritto, definita in uno smart contract che espone un'interfaccia standardizzata***. *La funzione degli standard di tokenizzazione è stabilire un insieme coerente di funzioni ed eventi per creare, trasferire e distruggere unità logiche, così che contratti diversi possano interoperare con gli stessi strumenti applicativi e infrastrutturali*.

La documentazione ufficiale di Ethereum colloca gli standard ERC tra le specifiche applicative che abilitano compatibilità, componibilità e verificabilità dei processi, grazie a interfacce comuni e a log di eventi che descrivono ogni mutazione di stato in modo osservabile e persistente [9]. **In questo quadro, la scelta dello standard determina la semantica del bene rappresentato e i vincoli del suo ciclo di vita, oltre a incidere sui costi operativi, sulla facilità di integrazione e sulla possibilità di ricostruire in modo affidabile la storia delle operazioni.** Tre famiglie di standard sono di interesse generale:

- **ERC-20** definisce un'API per token fungibili, cioè **quantità perfettamente intercambiabili dello stesso bene**. *Le primitive essenziali includono trasferimenti diretti, autorizzazioni alla spesa da parte di terzi e interrogazione dei saldi per indirizzo, accompagnate da eventi che registrano le variazioni in modo verificabile.* La finalità esplicita è garantire un'interoperabilità forte, per cui qualunque token conforme venga trattato in modo uniforme da wallet[7], exchange[8] e strumenti di analisi [10]. Gli impieghi più diffusi coincidono con scenari in cui la fungibilità è un requisito intrinseco: mezzi di scambio e riserva di valore, stablecoin[9] agganciate a unità di conto esterne e token di governance che rappresentano diritti di voto in sistemi decentralizzati. Esempi ormai consolidati includono USDC[10], documentato come

[7] software o dispositivo hardware, che consente di **gestire e utilizzare le chiavi crittografiche** necessarie per accedere ai propri asset digitali

[8] piattaforma digitale che consente di **comprare, vendere e scambiare criptovalute e token**, fungendo da mercato regolato o semi-regolato per gli asset digitali.

[9] criptovaluta progettata per mantenere **un valore stabile nel tempo**, generalmente ancorato a un asset di riferimento come una valuta fiat

[10] stablecoin ancorata al dollaro statunitense.

token nativo ERC-20 sulla rete Ethereum [17], e DAI [18]. In questi casi **l'informazione rilevante è essenzialmente quantitativa e ogni unità può sostituire qualsiasi altra senza perdere significato.**

- **ERC-721** adotta un paradigma diverso, fondato sull'unicità di un'istanza. Ogni identificatore rappresenta un bene non fungibile e lo standard specifica delle funzioni per gestire aspetti quali titolarità, trasferimenti e deleghe operative su singoli esemplari, oltre ad un modello di metadati per descriverne gli attributi [11]. **La semantica di ERC-721 è appropriata quando ciò che conta è l'identità del singolo asset e la sua tracciabilità come esemplare distinto**, per esempio per titoli di proprietà, certificati o oggetti digitali non sostituibili senza perdita informativa.

- **ERC-1155 generalizza le due prospettive precedenti e definisce una singola interfaccia per contratti in grado di rappresentare molteplici classi di asset, fungibili e non fungibili, identificate da un parametro id**. *Il contratto mantiene saldi per id e per account, espone trasferimenti sicuri in modalità singola e in modalità batch e registra tutte le operazioni con gli eventi TransferSingle e TransferBatch, così da ridurre costi e latenza quando si movimentano insiemi di asset eterogenei nella stessa transazione [12]. La specifica prevede inoltre un'interfaccia di metadati che, dato un id, restituisce un URI verso una descrizione esterna, consentendo di separare la prova di esistenza e di trasferimento, che resta on-chain, dalla descrizione ricca, che può essere mantenuta off-chain ma resa verificabile [13].* Questa combinazione di flessibilità, efficienza (operazioni batch) e indirizzabilità informativa (URI per id) è alla base dell'adozione di ERC-1155 in domini in cui coesistono quantità omogenee e identità distinte.

Un elemento trasversale a tutti gli standard è l'identificazione univoca degli attori che interagiscono con i token attraverso address univoci. *Ogni soggetto che detiene o movimenta token è individuato da un indirizzo sulla rete e i log degli eventi, generati secondo l'interfaccia standard, permettono di attribuire con precisione le operazioni alle controparti [9].* Questo aspetto è *determinante per l'analisi a posteriori e per i requisiti di audit*, perché **consente di ricostruire passaggi di responsabilità** e di verificare la coerenza tra stati attesi e stati on-chain. Inoltre, il fatto che le interfacce siano ampiamente documentate e stabili nel tempo facilita l'adozione di strumenti comuni per l'osservabilità e la riconciliazione dei dati, elementi indispensabili quando il dominio applicativo richiede garanzie sulla qualità e sull'integrità delle informazioni.

OpenZeppelin e Standard ERC

OpenZeppelin è una libreria open source che fornisce interfacce e implementazioni aggiornate degli standard ERC, unitamente a componenti ausiliari per esigenze non funzionali tipiche di ambienti produttivi. La libreria è **ampiamente adottata nell'industria** perché riduce il rischio applicativo **promuovendo la conformità alle specifiche Ethereum Improvement Proposal *(EIP)* e accelerando lo sviluppo mediante moduli riusabili sottoposti a manutenzione e revisione continue**. La documentazione copre sia gli aspetti d'interfaccia sia le linee guida per l'integrazione con i servizi dell'ecosistema, con l'obiettivo di fornire una base affidabile e allineata agli standard per l'implementazione di smart contract [14][19]. In questo contesto, ERC-20, ERC-721 ed ERC-1155 rappresentano tre punti di ancoraggio stabili, ognuno con una superficie d'API chiara e con un insieme di eventi che gli strumenti esterni si aspettano di osservare. Le Ethereum Improvement Proposal (EIP) sono il meccanismo attraverso cui la comunità Ethereum propone e discute aggiornamenti al protocollo Ethereum, gli Ethereum Request for Comments (ERC) sono una sottocategoria di EIP specificatamente dedicata alla definizione di interfacce per token e smart contract. Ogni ERC è quindi una EIP che specifica un insieme di funzioni ed eventi attesi, al fine di garantire compatibilità nell'ecosistema. Segue una comparazione dei principali standard ERC, riassunta nella *Fig. 6.*

Per ERC-20, OpenZeppelin espone contratti conformi a EIP-20 con le primitive attese di trasferimento, autorizzazione e interrogazione dei saldi, oltre agli eventi standard che i sistemi di terze parti utilizzano per indicizzare e riconciliare le operazioni [10][14]. L'obiettivo dello standard è l'interoperabilità piena: una superficie comune consente a wallet e exchange di trattare in modo uniforme token emessi da soggetti diversi, con un profilo di rischio ridotto e con processi di integrazione ripetibili.

Per ERC-721, la libreria fornisce un'implementazione che gestisce in modo rigoroso proprietà, trasferimento, approvazioni e metadati per asset non fungibili. La specifica EIP-721 chiarisce l'idoneità del modello alla rappresentazione di beni digitali o fisici nei quali l'identità individuale non può essere sostituita senza perdita informativa [11].

Per ERC-1155, OpenZeppelin implementa la semantica multi-token prevista da EIP-1155, includendo gli eventi TransferSingle e TransferBatch, le operazioni di trasferimento sicuro in modalità singola e in modalità batch e l'interfaccia dei metadati che associa a ogni id un URI [14]. La pagina di riferimento su ethereum.org sottolinea che un singolo contratto può includere combinazioni arbitrarie di token, con vantaggi in termini di costo e latenza quando la logica applicativa richiede di operare su insiemi di asset [13]. Un aspetto architetturale rilevante è la separazione tra evidenze on-chain e componenti descrittive off-chain. L'interfaccia dei metadati prevede che, per ogni id, il contratto esponga un URI. Questo approccio permette di ridurre i costi di memorizzazione on-chain mantenendo un alto grado di verificabilità e di immutabilità logica, a condizione che siano adottate politiche adeguate alla persistenza e alla disponibilità dei contenuti.

Caratteristica	ERC-20	ERC-721	ERC-1155
Tipo di asset	Fungibili (token identici e intercambiabili)	Non fungibili (ogni token è unico)	Multi-token: supporta sia fungibili che non fungibili nello stesso contratto
Esempi tipici	Stablecoin (USDT, USDC), utility token	NFT (arte digitale, collectibles)	Giochi (armi, skin, oggetti), marketplace multi-asset
Identificatore token	Solo balanceOf di un account	Ogni token ha un tokenId unico	Ogni tipo di asset ha un id, con quantità associata
Gestione quantità	Bilancio per indirizzo (balanceOf)	Ogni tokenId appartiene a un solo owner	Ogni id può avere più unità (semi-fungibili)
Efficienza gas	Alta efficienza, semplice	Costoso (ogni trasferimento è separato)	Più efficiente: batch transfer di più asset in una sola transazione
Compatibilità	Ampiamente supportato in DeFi e wallet	Supporto NFT (OpenSea, marketplace)	Supporto crescente in gaming e marketplace multi-asset
Standardizzazione	ERC-20 (2015) maturo	ERC-721 (2018) consolidato	ERC-1155 (2019) più flessibile e moderno

Figura 6: Tabella comparativa degli standard ERC

Standard ERC-1155 per la rappresentazione dei lotti

La rappresentazione dei lotti in una supply chain richiede di combinare quantità omogenee, identità di lotto e genealogie informative. In un modello puramente fungibile come ERC-20 non esiste distinzione per lotto, perché tutte le unità sono indistinguibili; in un modello puramente non fungibile come ERC-721 non esiste quantità per identificatore, perché ogni token è un esemplare unico. ERC-1155 colma esattamente questo spazio progettuale. Un id rappresenta una classe di asset con quantità variabile nel tempo e l'URI per id collega la classe a metadati strutturati e verificabili [12][13].

Questo consente di trattare un lotto come asset fungibile per id, mantenendo una scheda informativa stabile che descriva in modo completo il lotto stesso, i suoi attributi principali e gli eventuali riferimenti documentali. Le operazioni previste all'interno degli standard ERC, in particolare di ERC-1155, permettono di proiettare le attività astratte della filiera, cioè le operazioni previste dal modello Cerere, in modo diretto. La creazione di uno specifico lotto corrisponde per esempio all'operazione di mint di un token identificato da un id specifico; la cessione e l'acquisizione di lotti, o di parti di essi, corrispondono invece a trasferimenti *(di una determinata quantità di uno o più token, identificati dai loro id)* tra diversi address Ethereum che rappresentano gli attori coinvolti nell'operazione. Le trasformazioni sono invece esprimibili come combinazioni di burn di token *(id specifici e quantità a questi associate)* in input e mint di token *(id specifici e quantità a questi associate)* in output, con gli eventi TransferSingle e TransferBatch che fungono da traccia causale verificabile tra input e risultato [12][13].

Questa mappatura preserva l'atomiticità delle operazioni fondamentali e offre una base solida per ricostruire il percorso dei lotti lungo la catena, grazie a evidenze on-chain che non dipendono da un singolo archivio esterno.

L'uso di metadati per id consente di integrare nel modello informazioni che, pur non essendo economico conservare on-chain in forma estesa, risultano indispensabili per interpretare correttamente la transazione. L'adozione di *URI content-addressed*[11] ossia di URI basati sull'hash crittografico del contenuto a cui puntano, come nel caso di IPFS, permette di collegare la descrizione di un token ad un Content Identifier[12] (CID) calcolato crittograficamente; ciò garantisce che il riferimento ai metadati cambi solo se cambia il documento e che sia possibile verificare l'integrità del contenuto indipendentemente dal punto di pubblicazione [16]. In termini di architettura, la blockchain è impiegata quindi per registrare le transazioni che rappresentano le operazioni tra i lotti, mentre IPFS ospita i dati descrittivi che li documentano in modo strutturato. Entrambi i livelli restano verificabili e immutabili, ma la quantità di informazioni mantenute on-chain è ridotta al minimo necessario, contenendo i costi di memorizzazione. Questa separazione consente di preservare la trasparenza e la coerenza complessiva del sistema, in linea con gli obiettivi di interoperabilità e tracciabilità informativa promossi dal *Digital Product Passport*.

La scelta di ERC-1155 è motivata anche da considerazioni di efficienza operativa e di integrazione. Le filiere reali spesso richiedono operazioni simultanee su più lotti o su più classi di asset; la capacità dello standard di eseguire operazioni su insiemi di lotti, *cioè di operare contemporaneamente su un insieme di token id diversi in un'unica transazione,*

[11] Esempio di URI content-addressed IPFS: *ipfs://bafybeigdyrzt4s3j7zbyd7n3r2cz7x6t2c4a2fz4o7n4k7ogv5v7sh7cqa*

[12] Esempio di Content-Identifier, parte dell'URI sopra riportato: *bafybeigdyrzt4s3j7zbyd7n3r2cz7x6t2c4a2fz4o7n4k7ogv5v7sh7cqa*

riduce la latenza e i costi transazionali rispetto a sequenze di operazioni elementari, preservando al tempo stesso la coerenza globale del sistema [12][13]. L'identificazione univoca degli attori tramite address consente di associare ogni movimento a un soggetto chiaramente individuato, facilitando audit e riconciliazioni. Inoltre, la disponibilità di implementazioni consolidate come quelle fornite da OpenZeppelin attenua il rischio di deviazioni dallo standard, facilitando l'evoluzione del codice e garantendo compatibilità con gli strumenti dell'ecosistema.

Transazioni ed Eventi

In Ethereum, una **transazione** costituisce l'unità atomica di mutazione dello stato globale. Essa è definita come un messaggio firmato crittograficamente, costruito localmente da un utente e propagato alla rete affinché venga verificato, incluso in un blocco ed eseguito dalla Ethereum Virtual Machine. La transazione non modifica direttamente dati preesistenti, ma produce una **transizione di stato**: lo stato corrente della blockchain è infatti il risultato deterministico dell'applicazione ordinata di tutte le transazioni validate. L'immutabilità del registro si riferisce alla sequenza delle transizioni registrate, che una volta incluse in blocchi confermati non può essere alterata retroattivamente.

Campi transazione	Descrizione
chainId	ID della rete (es. 1 = Mainnet)
nonce	Contatore transazioni del mittente
maxPriorityFeePerGas	Mancia (in wei per unità di gas)
maxFeePerGas	Fee massima per unità di gas
gasLimit	Unità di gas massime consentite
to	Destinatario (20 byte, vuoto = creazione contratto)
value	ETH da inviare (in wei)
data	Payload: bytecode contratto / chiamata funzione / vuoto
accessList	Opzionale: indirizzi pre-dichiarati + slot di storage
v	ID di recupero della firma
r	Firma (parte 1)
s	Firma (parte 2)

Figura 7: Principali parametri di una transazione ETH

Ogni transazione *(Fig. 7)* include campi obbligatori: l'indirizzo del destinatario (*to*, che può corrispondere a un contratto), un **nonce** progressivo per prevenire riutilizzi e garantire ordinamento, i parametri relativi al gas, un eventuale valore monetario in ether, e il campo **data** che incapsula i parametri di invocazione delle funzioni contrattuali. La firma digitale, calcolata mediante ECDSA sulla chiave privata dell'utente, garantisce autenticità e integrità, consentendo a ogni nodo della rete di verificare la provenienza e la correttezza del messaggio tramite la corrispondente chiave pubblica *(Fig. 8)*. Gli **eventi** sono invece dei log strutturati che uno smart contract emette durante l'esecuzione di una transazione e che vengono inclusi nella relativa ricevuta. Essi non determinano modifiche di stato, ma descrivono in forma sintetica e standardizzata le operazioni effettuate, costituendo un meccanismo di osservabilità. I log generati sono permanenti, indicizzabili e verificabili, e rappresentano la fonte informativa primaria per applicazioni esterne quali wallet, sistemi di analisi e servizi di audit. Nel caso di ERC-1155, gli eventi TransferSingle e TransferBatch forniscono informazioni sufficienti a ricostruire con completezza i trasferimenti, rendendo superflua l'ispezione diretta delle variabili interne del contratto.

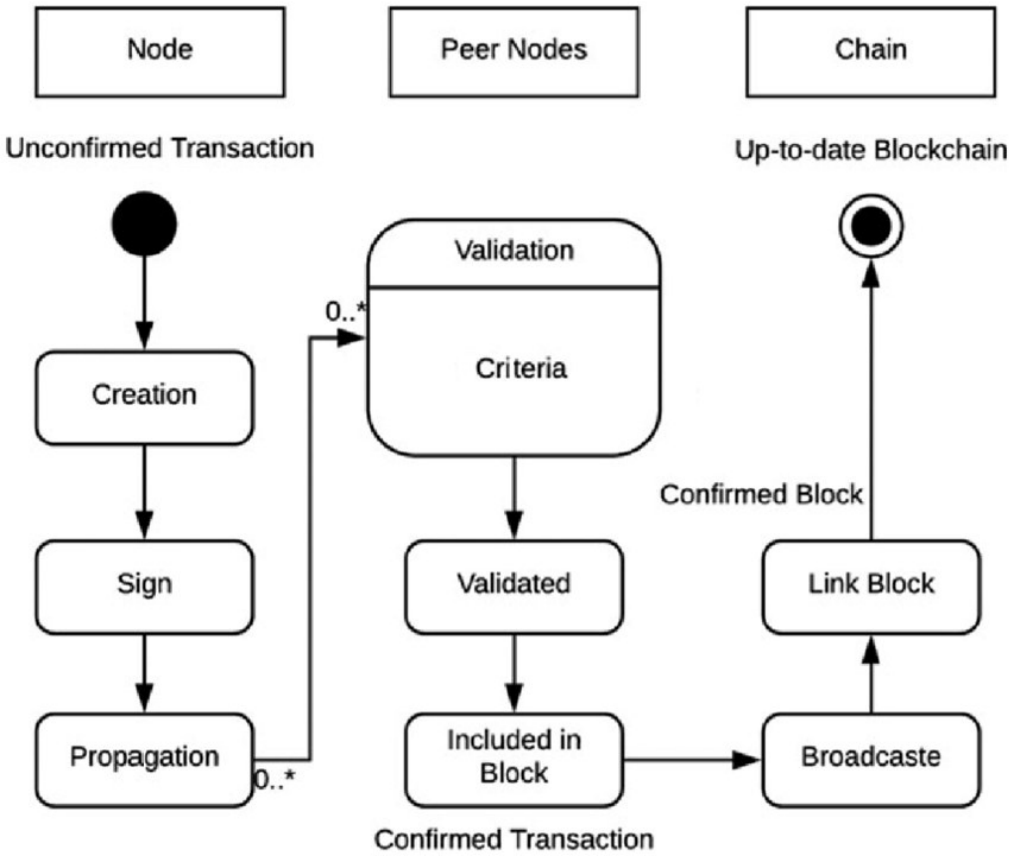


Figura 8: Flusso di una transazione ETH

Nell'architettura proposta, gli attori della supply chain sono identificati da **address** sulla rete Ethereum e interagiscono attraverso le chiavi crittografiche con cui firmano le transazioni. Ogni **token ERC-1155** rappresenta un **lotto specifico**, ed è modellato come un identificatore associato a una quantità fungibile di unità. Le unità costitutive del lotto possono essere create (*mint*), trasferite (*transfer*) o eliminate (*burn*), in corrispondenza con le operazioni astratte del modello Cerere. La dinamica della filiera è dunque interamente gestita on-chain: la semantica delle operazioni è incorporata nello standard, e la tracciabilità è garantita dagli eventi immutabilmente registrati nel registro distribuito.

Alla creazione di un token ERC-1155 viene associato, allo specifico id, un **URI** che punta a un documento JSON conservato su **IPFS**. Tale documento raccoglie i **metadati** del lotto, comprendenti attributi descrittivi e riferimenti documentali. L'uso di un sistema content-addressed garantisce che l'identificatore del documento cambi esclusivamente in caso di variazione del contenuto, assicurandone così la verificabilità e l'immutabilità logica. L'architettura separa quindi le responsabilità: da un lato la blockchain conserva la prova di esistenza, di titolarità e di movimentazione; dall'altro IPFS custodisce la descrizione estesa, che resta off-chain ma convalidabile crittograficamente. Per esigenze applicative, i dati on-chain e i metadati off-chain vengono ulteriormente aggregati in un **database ausiliario** (MongoDB), che non sostituisce le evidenze primarie, ma consente interrogazioni e visualizzazioni efficienti. Questa distinzione sarà approfondita nei capitoli successivi, dove verranno illustrate la modellazione dei lotti in termini di struttura JSON e la corrispondenza sistematica tra operazioni del modello Cerere e primitive ERC-1155 (*mint*, *transfer*, *burn*).

Capitolo 4: Analisi dei requisiti, modello dei dati

Per poter realizzare un sistema efficace è necessario definire un modello dei dati chiaro e coerente, che stabilisca quali informazioni devono essere conservate, in che forma e su quale livello tecnologico. Partendo dall'idea di rappresentare i lotti della supply chain come token ERC1155, in modo da sfruttare le operazioni native offerte dallo standard, si procede alla modellizzazione della struttura dei dati associati a un lotto. Ogni lotto è descritto attraverso un insieme di elementi fondamentali: l'identificativo univoco, i dati di creazione che ne attestano l'origine e il produttore, i dati di tracciabilità che ne registrano le evoluzioni e i trasferimenti, e infine gli allegati che completano la documentazione (come certificati di qualità o analisi di laboratorio). Questi dati sono organizzati in una struttura JSON che viene illustrata e commentata, mettendo in evidenza il significato di ciascuna sezione, il contenuto che ospita e la modalità di gestione scelta: database off-chain (MongoDB) per la parte di tracciabilità dinamica, IPFS per la conservazione dei metadati e degli allegati, blockchain per gli aspetti di proprietà, quantità e immutabilità. Dopo la descrizione della struttura di un lotto, il capitolo affronta la questione della mappatura tra operazioni astratte e operazioni sui token. Viene mostrato come le azioni di business tipiche della supply chain possano essere ricondotte alle primitive messe a disposizione dall'ERC1155, talvolta in combinazione, e corredate da eventi. Infine, viene presentato un modello complessivo che integra i diversi livelli tecnologici e le varie tipologie di dati, offrendo una visione unitaria del sistema.

Struttura di un lotto

Il modello proposto per la rappresentazione dei lotti si concretizza in un documento JSON persistito in MongoDB. Tale documento nasce dall'aggregazione di tre domini informativi distinti: i dati pubblici conservati in IPFS, le informazioni crittograficamente garantite dalla blockchain e i dati operativi raccolti off-chain. Questa architettura permette di integrare livelli tecnologici eterogenei, mantenendo un riferimento coerente e consultabile per ogni lotto. La struttura si articola in due macro-componenti: **PublicData**, che raccoglie dati descrittivi statici e riferimenti documentali destinati alla consultazione pubblica e **TraceabilityData**, che include informazioni dinamiche di stato, prove blockchain e cronologia delle operazioni. Il flusso operativo segue uno schema ben definito: inizialmente viene generato un documento JSON contenente i dati del lotto. Questo file viene caricato su **IPFS**, che restituisce un identificativo univoco (hash/CID). L'URI risultante diventa il riferimento immutabile e decentralizzato del lotto e viene utilizzato nello smart contract per effettuare l'operazione di **mint** di un token ERC-1155 con un id specifico. In questo modo, ogni lotto acquisisce una rappresentazione on-chain ancorata al suo JSON su IPFS. Le successive operazioni di business come trasferimenti, split o merge... vengono registrate come eventi o transazioni ERC-1155 per l'id specifico e riflessi nella sezione **TraceabilityData**, che mantiene la memoria storica dello stato del lotto e ne consente la consultazione off-chain tramite MongoDB. Di seguito si riporta una semplificazione delle sezioni per illustrare le principali entità coinvolte. La descrizione della struttura di un lotto proposta non ha l'obiettivo di analizzare nel dettaglio ogni singolo campo, bensì di chiarire quali concetti fondamentali vengono gestiti: che tipo di prodotto esso rappresenti, chi lo abbia creato, in quale contesto e con quali modalità.

Identificatore principale

Il batchId, coincidente con il tokenId ERC-1155, rappresenta l'identificativo univoco del lotto.


```
{
  "batchId": 1
}
```


PublicData

Questa sezione raccoglie i dati quantitativi del lotto, l'attore creatore, l'operazione di origine, le informazioni di prodotto, le relazioni genealogiche (parentBatchesIDs*, cioè gli id dei lotti direttamente coinvolti nella generazione del lotto considerato)* e gli eventuali allegati documentali caricati su IPFS. Tutti questi elementi costituiscono la dimensione descrittiva del lotto, immutabile e pubblicamente accessibile.


```
{
  "publicData": {
    "quantity": 1000,
    "unitOfMeasure": "kg",
    "productionDate": "2024-06-01",
    "batchCreator": {
      "address": "0x742d35Cc6634C0532925a3b844Bc454e4438f44e",
      "name": "Raw Materials Inc.",
      "role": "producer"
    },
    "operation": {
      "operationId": "op-123",
      "operationType": "creation",
      "location": "Factory A"
    },
    "product": {
      "productId": "RAW-001",
      "name": "Raw Material A",
      "category": "RAW_MATERIAL"
    },
    "parentBatchesIds": [],
    "attachments": [
      {
        "attachmentId": "att-123",
        "ipfsUri": "ipfs://QmXyZ...",
        "description": "Certificate of Analysis",
        "fileName": "coa.pdf"
      }
    ]
  }
}
```

TraceabilityData

Questa parte contiene i riferimenti ai metadati IPFS, le prove blockchain della creazione del token (txhash, blocco, timestamp), lo stato del lotto definito dai campi lifecycleStatus *(inutilizzato, parzialmente o interamente consumato, parzialmente o completamente distrutto)* e contaminationStatus (GREEN / YELLOW / RED), la quantità residua, la cronologia dei trasferimenti[13] e le operazioni atomiche tokenizzate. In questo modo, ogni evento di business trova un corrispettivo registrato on-chain e un riflesso operativo off-chain.

```
"traceabilityData": {
  "batchIpfsUrl": "ipfs://QmXXX...",
  "batchIpfsHash": "QmXXX...",
  "txHash": "0x123...",
  "blockNumber": 5,
  "blockTimestamp": 1640995200,
  "contaminationStatus": "GREEN",
  "lifecycleStatus": "UNUSED",
  "currentQuantity": 1000,
  "transfers": [...],
  "atomicTokenizedOperations": [{
    "tokenizedOpType": "mint",
    "operationId": "op-123",
    ...
  }]
}
```

L'analisi della popolazione dei campi sarà affrontata in seguito, attraverso lo studio del comportamento degli endpoint applicativi. A questo punto diviene invece centrale introdurre le operazioni sui token ERC-1155 e la loro mappatura rispetto alle tipiche azioni della supply chain; ciò, congiuntamente al modello di funzionamento delle transazioni Ethereum già delineato, consente di presentare un modello dei dati capace di aggregare le diverse entità coinvolte in termini astratti, fornendo una visione d'insieme propedeutica all'introduzione dell'architettura del sistema.

[13] Il campo transfers contiene le operazioni di trasferimento, totali o parziali, relative al lotto descritto, indicando quantità, mittente, destinatario e momento della transazione, in coerenza con i dati on-chain.

Mappatura tra operazioni astratte e operazioni tokenizzate

Lo standard ERC-1155, come implementato nella libreria OpenZeppelin, fornisce un insieme di operazioni fondamentali per la gestione di asset digitali eterogenei all'interno di un unico smart contract. Le primitive principali sono:

- **Mint** crea nuovi token introducendo una certa quantità di asset sulla blockchain.
- **Burn** distrugge o consuma una quantità di token esistenti riducendone la disponibilità.
- **SafeTransferFrom** e **SafeBatchTransferFrom** trasferiscono *in modo sicuro*[14] uno o più token da un indirizzo a un altro verificando la corretta ricezione da parte del destinatario.

Completano il modello funzionale gli eventi standard che consentono di tracciare le operazioni eseguite on-chain. Nel nostro sistema di tracciabilità queste primitive rappresentano i mattoni elementari attraverso cui vengono modellizzate le operazioni di business tipiche della supply chain *(Fig. 9)*. In particolare, in riferimento al modello Cerere. È opportuno osservare che l'adattamento del modello alle primitive ERC-1155 comporta alcune differenze concettuali: le attività di Acquisition e Providing trattate come momenti distinti nel modello originario sono state uniformate in un'unica operazione di transfer. Sul piano tecnico, infatti, il trasferimento di un token ERC-1155 implica simultaneamente la cessione da parte di un attore e l'acquisizione da parte di un altro rendendo superflua la distinzione. L'unica eccezione è rappresentata dall'acquisition dalla natura, cioè la generazione iniziale di un lotto, che rimane modellizzata come operazione di mint. Una razionalizzazione analoga riguarda la categoria di Alteration. Nel nostro approccio tutte le modifiche sostanziali che comportano un cambiamento di identità del lotto sono trattate come trasformazioni implementate quindi come combinazioni di burn e mint. L'unico attributo previsto come alterabile senza generare un nuovo lotto è lo stato di contaminazione ma questo non fa parte delle primitive standard ERC-1155 ed è stato introdotto come estensione applicativa costituendo un arricchimento specifico rispetto alla semantica originaria.

[14] Il termine "sicuro" si riferisce al controllo previsto dallo standard ERC-1155 e implementato da OpenZeppelin: se il destinatario è un contratto, viene verificato che implementi IERC1155Receiver per poter ricevere i token; se invece è un EOA (Externally Owned Account, cioè un normale indirizzo utente privo di codice), il trasferimento viene accettato direttamente poiché non necessita di verifica.

Attività Cerere	Caso d'uso nel sistema	Operazioni Token
Acquisition	Introduzione di un nuovo lotto	`mint()`
Acquisition / Providing	Cessione / acquisizione tra attori	`transfer()`
Division	Split di un lotto in N lotti	`burn() + mint()`
Integration	Merge di più lotti in uno	`burn() + mint()`
Alteration (stato)	Cambio di stato del lotto	`STATUS_CHANGE (evento custom)`
Alteration (trasformativa)	Modifica con cambio identità del lotto	`burn() + mint()`
Transformation	Trasformazione di lotti in nuovo lotto	`burn() + mint()`

Figura 9: Mappatura operazioni

A questo stadio dell'analisi disponiamo di una comprensione del funzionamento delle transazioni su blockchain e abbiamo delineato l'impianto concettuale del sistema. In termini operativi, il modello si articola come segue: ciascun lotto è rappresentato da un **token** a cui è associato un identificativo univoco e che rimanda a un documento JSON immutabile conservato su **IPFS**. Le operazioni di business che interessano i lotti si traducono in corrispondenti **operazioni sui token** (mint, burn, transfer, nonché loro combinazioni per split e merge), implementate come transazioni Ethereum registrate on-chain. Gli **attori** che prendono parte a tali processi sono identificati dal proprio indirizzo e garantiscono la validità delle interazioni mediante la firma crittografica con la rispettiva chiave privata. Su queste basi è ora possibile introdurre un **modello dei dati** che sintetizza il contesto complessivo e ne rappresenta le entità fondamentali, le relazioni e le proprietà rilevanti.

Definizione del modello dei dati

Per rappresentare in maniera sintetica e coerente le entità del dominio e le relazioni che le legano, è stata elaborata una astrazione concettuale sotto forma di diagramma ER. Lo scopo di questo modello non è mostrare dove o come i dati vengano memorizzati, ma offrire una visione generale dei concetti fondamentali e delle loro interconnessioni, fornendo così una base per comprendere il funzionamento complessivo del sistema prima di affrontare i dettagli implementativi.

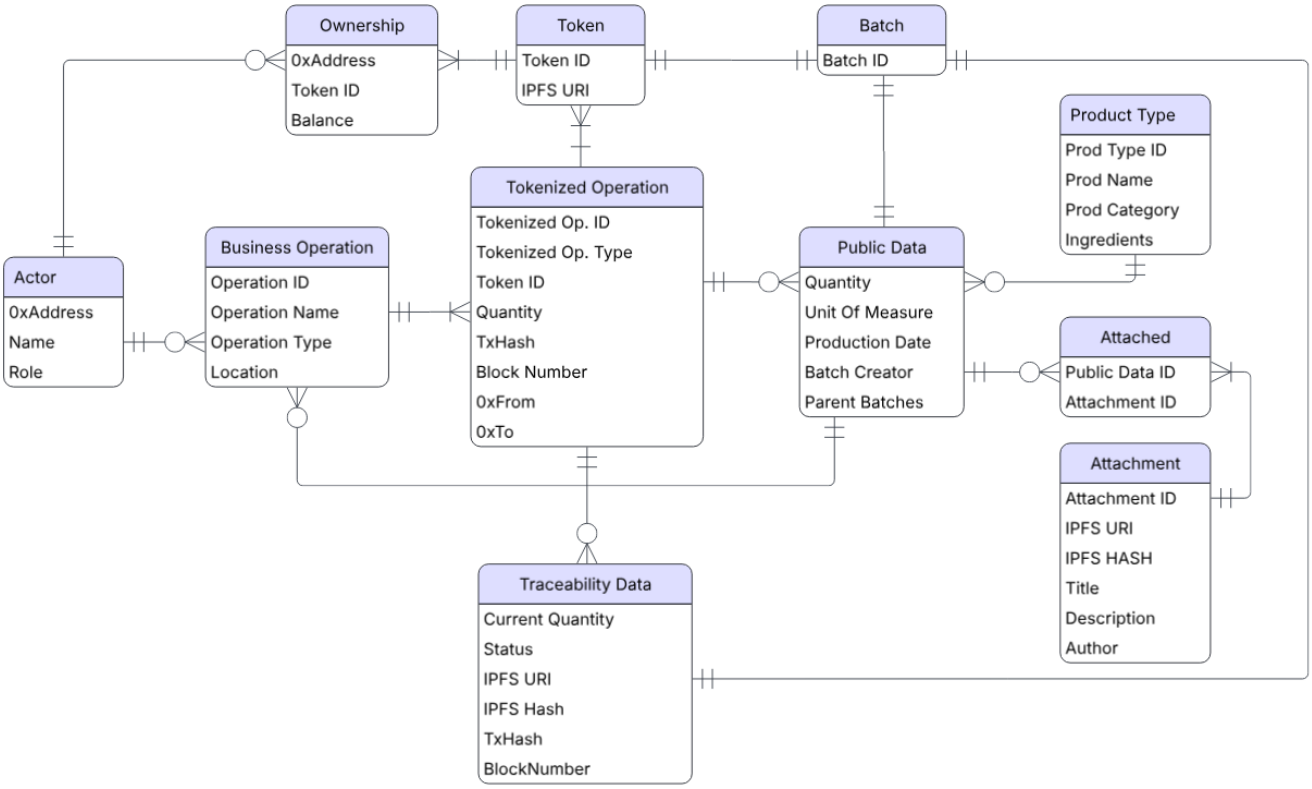


Figura 10: Modello dei dati

L'entità principale del modello è il **Batch**, che rappresenta l'unità logica aggregata di tracciabilità. *Ogni batch è identificato da un ID univoco che corrisponde 1:1 al tokenId ERC1155 associato*, fungendo da punto di ancoraggio tra la dimensione logica del lotto e la sua rappresentazione digitale sulla blockchain. Al batch sono collegati i **dati pubblici**, che descrivono le condizioni di creazione o generazione del lotto: *quantità iniziale, unità di misura, data di produzione, attore creatore, tipologia di prodotto, batch genitori e allegati documentali*. Questi ultimi consentono di arricchire la descrizione del lotto con certificazioni, analisi di laboratorio o altri riferimenti caricati su IPFS. **L'insieme dei dati pubblici costituisce la parte immutabile e descrittiva del lotto ed è reso disponibile attraverso l'URI registrato nello smart contract per il tokenid specifico, che punta al documento JSON conservato su IPFS.** Accanto a essi si collocano i **dati di tracciabilità**, che *documentano l'evoluzione del lotto nel tempo e includono lo storico delle transazioni riferite al tokenId specifico*. In questo modo il batch è definito dall'identificativo, dai dati di creazione e dallo storico delle operazioni che ne descrivono l'intero ciclo di vita. Le operazioni

considerate nel modello sono le **operazioni di business** tipiche della supply chain, come descritte nel modello Cerere. Esse sono caratterizzate da un insieme minimo di attributi quali identificativo, tipologia, denominazione e luogo di esecuzione, e rappresentano i processi aziendali. La loro esecuzione coinvolge la rappresentazione digitale dei lotti e si traduce in **operazioni tokenizzate**, cioè primitive ERC1155 che realizzano concretamente le trasformazioni di mint, burn, transfer o combinazioni di esse. I dati di tracciabilità conservano la registrazione di queste operazioni, permettendo di ricostruire con precisione la sequenza di eventi che hanno modificato lo stato del lotto.

Capitolo 5: Design dell'architettura del sistema

Il modello dei dati definito nei capitoli precedenti rappresenta la base concettuale su cui si innesta la descrizione degli strumenti e delle tecnologie adottate nel sistema. A partire da tale rappresentazione è infatti possibile introdurre i componenti utilizzati per l'implementazione: Ganache ed Ethereum per la gestione della blockchain, MongoDB per la persistenza dei dati off-chain, e IPFS con il servizio Pinata per l'archiviazione decentralizzata di metadati e degli allegati associati ai lotti. Su queste tecnologie si appoggiano le API REST, sviluppate in Node.js con Express, che forniscono l'interfaccia HTTP di interoperabilità ed espongono le operazioni di business rispetto alla logica sottostante. Il tutto converge nell'architettura core del sistema, basata su un modello a tre livelli di storage che integra blockchain, database off-chain e storage distribuito. In questa sezione vengono presentati i principali componenti, suddivisi per livello, evidenziando le motivazioni che hanno guidato le scelte e le implicazioni derivanti.

Tecnologie e strumenti adottati

Il sistema di tracciabilità è stato progettato secondo un'architettura a più livelli, in cui ciascun componente tecnologico contribuisce a soddisfare esigenze specifiche. In questa sezione vengono presentati gli strumenti principali impiegati, suddivisi in base al livello architetturale di appartenenza. Si analizza dapprima la componente blockchain, successivamente il backend con le relative API, e infine i servizi di storage off-chain.

Blockchain

L'implementazione della componente blockchain si basa su un ecosistema Ethereum simulato attraverso Ganache, che fornisce un ambiente di sviluppo controllato e riproducibile *(Fig. 11)*. La scelta di Ganache è motivata dalla necessità di disporre di un ambiente deterministico durante le fasi di sviluppo e testing, eliminando le variabilità intrinseche delle reti pubbliche e riducendo i costi associati alle transazioni di test. Gli smart contract sono stati implementati in Solidity, linguaggio che rappresenta il de facto standard per lo sviluppo di contratti intelligenti su piattaforme compatibili con l'Ethereum Virtual Machine. L'adozione delle librerie OpenZeppelin costituisce una scelta strategica volta a minimizzare i rischi attraverso l'utilizzo di implementazioni battle-tested degli standard ERC. Il processo di deployment è orchestrato attraverso Hardhat, l'interazione runtime con la blockchain è gestita tramite Ethers.js, libreria che offre un'astrazione JavaScript per le operazioni Web3[15].

[15] Per operazioni Web3 si intendono le interazioni tra un'applicazione e la blockchain, quali la lettura di dati on-chain, l'invio di transazioni firmate e la chiamata di funzioni degli smart contract.

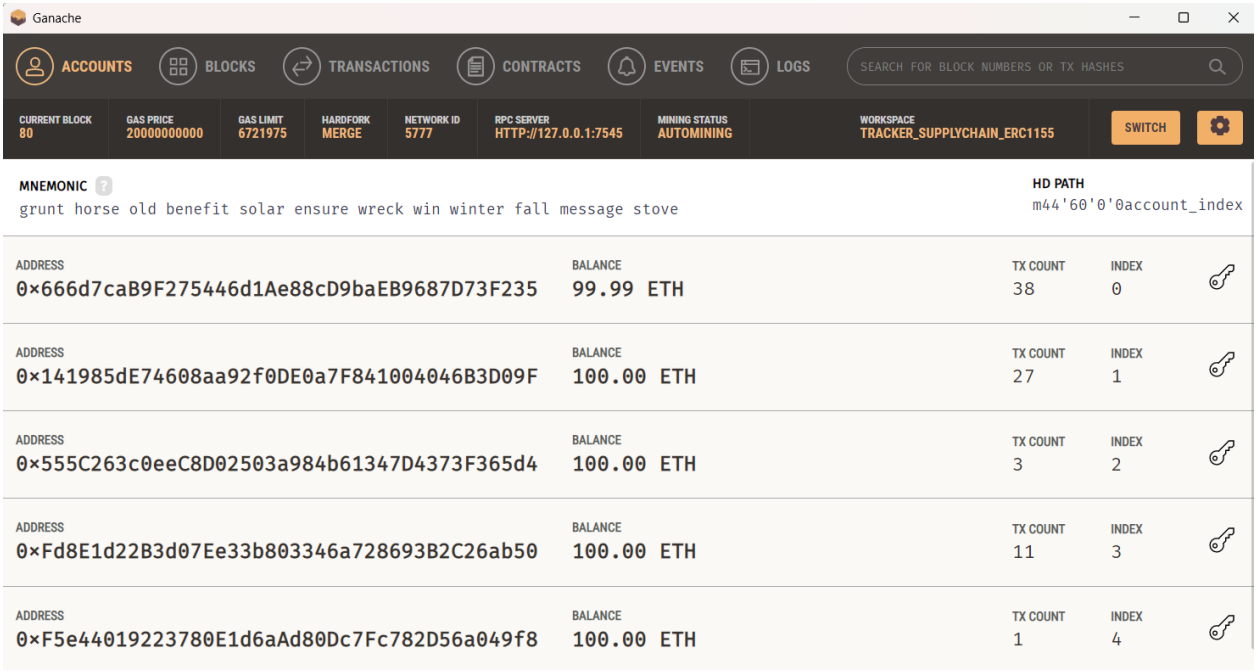


Figura 11: UI Ganache

Backend e API

Il backend applicativo è implementato in Node.js, scelta mirata alla conservazione di una coerenza tecnologica in ecosistema JavaScript. Express.js costituisce il framework di riferimento per l'esposizione delle API REST, gestisce il flusso delle richieste HTTP attraverso meccanismi di routing e middleware, ossia funzioni intermedie che elaborano i dati in ingresso come parsing e validazione prima della generazione automatica delle risposte in formato JSON. L'architettura modulare di Express permette di strutturare il codice secondo principi di separation of concerns, facilitando la manutenzione e l'estensibilità del sistema. Le interazioni con i servizi esterni sono mediate attraverso Axios, client HTTP che offre funzionalità per la manipolazione di richieste e risposte. Questa scelta architetturale centralizza la logica di comunicazione verso servizi esterni, semplificando la gestione di error handling.

Storage Off-Chain

L'adozione di IPFS (InterPlanetary File System) per lo storage distribuito è volta a garantire l'immutabilità e la verificabilità attraverso meccanismi di content-addressing. Nel contesto del sistema di tracciabilità, IPFS viene utilizzato per l'archiviazione permanente di metadati e documenti associati a ciascun token, con gli hash IPFS incorporati direttamente negli smart contract per garantire il collegamento immutabile tra identità on-chain e contenuti off-chain. L'integrazione con Pinata *(Fig. 12)* risolve le problematiche di persistenza e disponibilità di IPFS, fornendo servizi di pinning e API semplificate per l'upload e il retrieval dei contenuti. MongoDB è stato selezionato come sistema di persistenza off-chain, la natura document-oriented del database si allinea naturalmente con la struttura dei dati JSON

tipicamente gestiti in applicazioni Web3. Le funzionalità di MongoDB permettono di implementare query per l'analisi dei dati provenienti da fonti eterogenee.

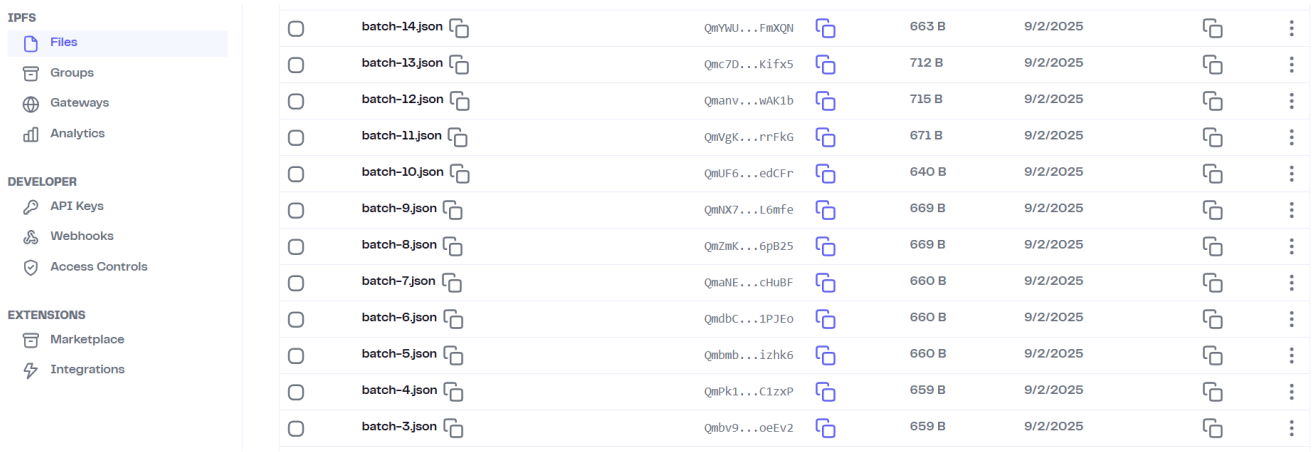


Figura 12: UI PInata

I livelli tecnologici descritti costituiscono la base del sistema, ma per poterli rendere fruibili da applicazioni client ed eventuali sistemi terzi è necessario un livello di interfaccia. Questo ruolo è assolto dalle API RESTful, che rappresentano il punto di accesso unificato al sistema.

API RESTful come interfaccia di interoperabilità

Le API RESTful rappresentano il livello di interfaccia attraverso il quale i client accedono alle funzionalità del sistema, garantendo un punto di ingresso uniforme e standardizzato. Basate sullo stile architetturale REST (Representational State Transfer), esse permettono di progettare servizi interoperabili secondo principi consolidati e semplificando l'integrazione con applicazioni eterogenee.

La progettazione di un'API RESTful si fonda sull'idea che ogni risorsa del dominio applicativo debba essere identificata in maniera univoca da un URI e resa accessibile attraverso operazioni standardizzate basate sul protocollo HTTP. Le azioni consentite seguono una semantica precisa: GET per interrogare lo stato di una risorsa, POST per la creazione, PUT o PATCH per la modifica e DELETE per la rimozione. Questa impostazione conferisce alle interfacce un carattere intuitivo e autoesplicativo, riducendo la necessità di conoscenze specifiche sul funzionamento interno del sistema e favorendo l'adozione da parte di applicazioni esterne. I dati vengono scambiati in formato JSON, che per la sua leggerezza e diffusione costituisce lo standard di fatto nell'implementazione di API moderne.

Operazione	Metodo	URI	Struttura semantica dell'URI
Creazione lotto	POST	/supply-chain/batches	Operazione sulla collezione batches creazione di una nuova risorsa
Trasferimento	POST	/supply-chain/batches/:batchId/transfer	Parametro :batchId per identificare il lotto segmento transfer per indicare l'azione
Consultazione	GET	/supply-chain/batches/:batchId	Metodo GET su URI parametrico accesso diretto alla risorsa identificata da :batchId

Figura 13: Alcuni esempi di URI esposti

Nel sistema sviluppato, gli URI *(Fig. 13)* sono stati progettati con un approccio semantico che rende immediatamente comprensibile la risorsa a cui si riferiscono e l'azione da eseguire. Tutti gli endpoint di interazione con i lotti fanno capo al prefisso comune "/supply-chain/batches", che identifica la risorsa principale del dominio, mentre l'aggiunta di segmenti o parametri consente di specializzare l'operazione. La coerenza di questa costruzione è evidente osservando alcuni esempi rappresentativi. La creazione di un lotto avviene tramite "POST /supply-chain/batches": l'uso del metodo POST indica la generazione di una nuova risorsa all'interno della collezione batches. Il trasferimento della proprietà di un lotto specifico è realizzato con "POST /supply-chain/batches/:batchId/transfer", dove il parametro :batchId identifica un lotto e il segmento transfer esplicita l'operazione richiesta. La consultazione dei dati relativi a un lotto è implementata come "GET /supply-chain/batches/:batchId", in cui il metodo GET comunica l'intenzione di recuperare informazioni senza modificarle e l'identificativo nel percorso consente un accesso puntuale alla risorsa. Il verbo HTTP definisce la natura dell'operazione, il percorso identifica la risorsa

e i segmenti aggiuntivi chiariscono l'azione specifica. In tal modo, l'interfaccia risulta coerente, leggibile e conforme ai principi REST, riducendo la necessità di documentazione accessoria e facilitando l'adozione da parte di sistemi esterni.

Architettura core del sistema

La rappresentazione riportata (*Fig. 14)* sintetizza la struttura complessiva del sistema, ponendo in evidenza le principali componenti infrastrutturali e la loro collocazione rispetto ai diversi livelli architetturali.

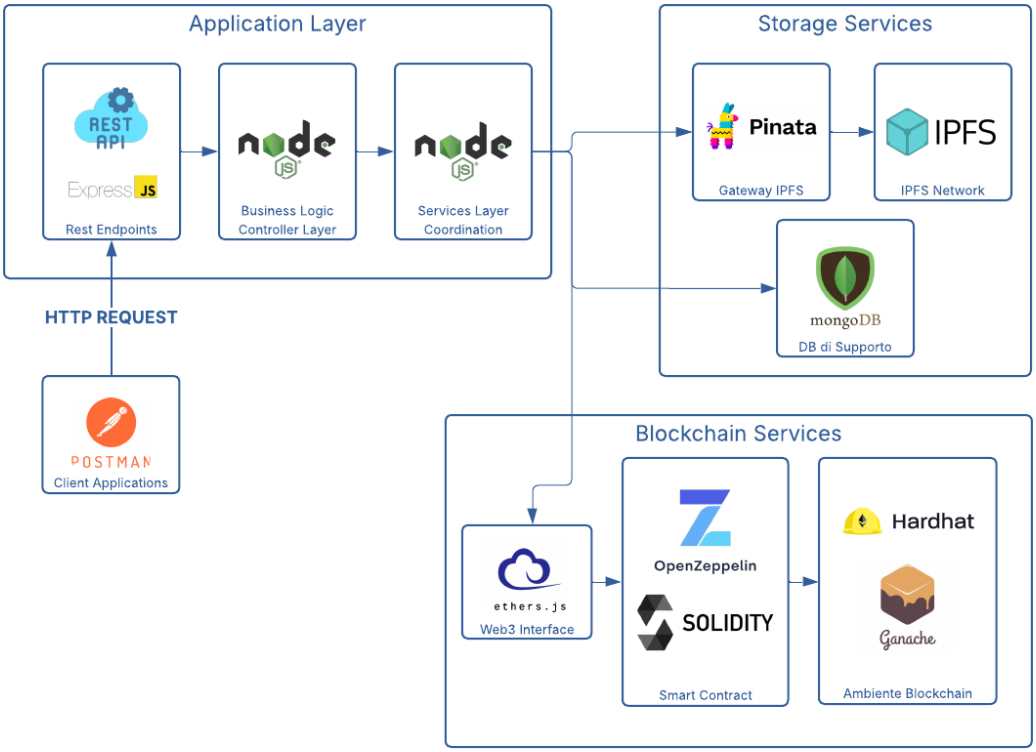


Figura 14: Architettura del Sistema

Il livello più esterno è costituito dalle **applicazioni client**, che interagiscono con il sistema attraverso richieste HTTP indirizzate all'interfaccia REST. A questo livello appartengono sia strumenti di testing *(ad esempio, Postman)* sia eventuali applicazioni destinate al consumo dei servizi esposti.

Il **layer applicativo** è strutturato secondo una suddivisione interna in tre sottocomponenti:

- gli **endpoint REST** realizzati con Express.js, che ricevono le richieste e instradano il flusso verso la logica interna;
- il **controller layer**, che incapsula la business logic e applica le regole operative del dominio;
- il **services layer**, che coordina l'accesso ai moduli esterni, mantenendo separazione tra logica applicativa e interazioni infrastrutturali.

Da questo livello si diramano le connessioni verso due insiemi di servizi specializzati. **Servizi blockchain**: l'interazione è mediata da **Ethers.js**, che fornisce l'astrazione Web3 necessaria a invocare le funzioni contrattuali e gestire eventi. Lo **smart contract**, implementato in Solidity sulla base delle librerie OpenZeppelin, costituisce il nucleo computazionale del sistema ed è eseguito in ambiente simulato tramite **Ganache** e orchestrato in fase di sviluppo con **Hardhat**. Per quanto concerne invece i **servizi di storage**, due componenti cooperano per garantire persistenza e verificabilità: **MongoDB**, impiegato per la gestione dei dati off-chain e per le query applicative, e **IPFS**, integrato con il servizio di pinning **Pinata**, utilizzato per l'archiviazione decentralizzata e content-addressed di metadati e allegati. L'articolazione delle componenti applicative e infrastrutturali mostra come il sistema sia progettato per integrare livelli eterogenei in un quadro coerente. Questa organizzazione trova il suo fondamento in un modello di persistenza a tre livelli che combina immutabilità e operazioni atomiche (blockchain), descrittività e contenuto (IPFS) ed efficienza di consultazione (MongoDB). In questo modo, ogni lotto mantiene un'identità univoca, una scheda informativa immutabile e una cronologia operativa interrogabile, elementi che verranno approfonditi nella sezione successiva.

Modello a tre livelli di storage: Blockchain, IPFS e MongoDB

Come mostrato nella *Fig. 14*, l'architettura del sistema integra diversi livelli, tra cui un insieme di servizi dedicati alla persistenza dei dati. A partire da questa visione complessiva, questa sezione illustra il modello a tre livelli di storage, illustrato nella *Fig. 15*, che descrive la cooperazione tra Blockchain, IPFS e MongoDB nella gestione delle informazioni on-chain e off-chain.

La persistenza dei dati è stata progettata secondo un modello a tre livelli, in cui **Blockchain**, **IPFS** e **MongoDB** svolgono ruoli complementari. Ogni livello è responsabile di una specifica dimensione informativa: la blockchain assicura immutabilità e titolarità delle operazioni, IPFS custodisce i contenuti descrittivi e documentali dei lotti, mentre MongoDB funge da aggregatore, integrando dati on-chain, off-chain e metadati applicativi per consentire interrogazioni ed elaborazioni efficienti. Questa visione consente di comprendere come le informazioni vengano distribuite e collocate nei diversi supporti tecnologici, prima di affrontare, nel capitolo successivo, la dinamica dei flussi che ne governano lettura e scrittura.

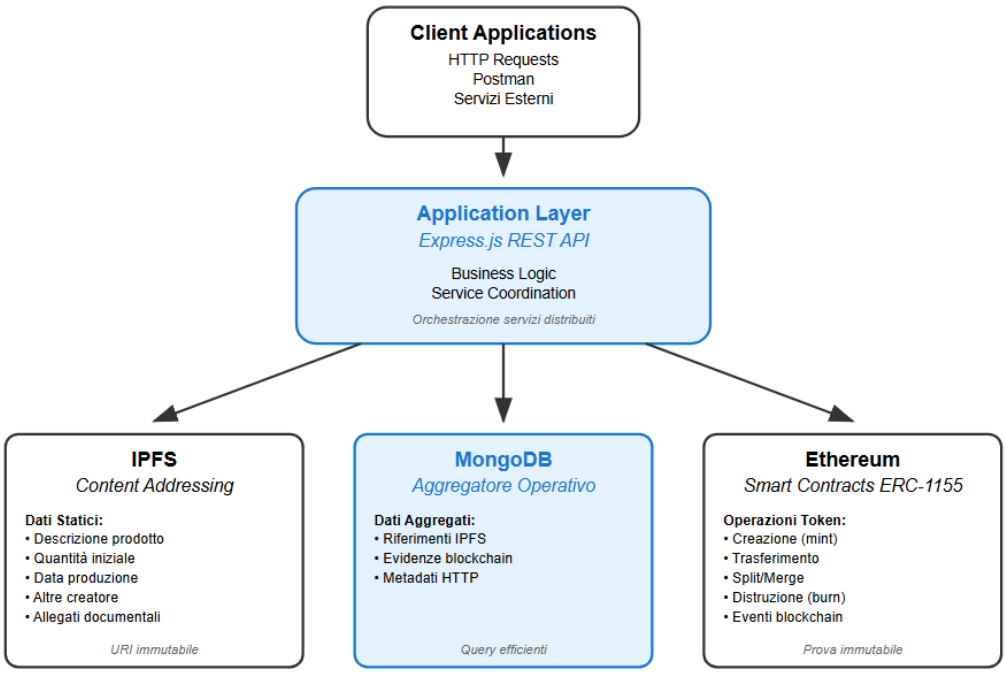


Figura 15: Architettura a tre livelli di Storage

- **IPFS**: ospita i dati pubblici associati alla creazione di un lotto (public data). Questi includono descrizione, quantità iniziale, data di produzione, attore creatore e allegati documentali. L'URI generato dal caricamento diventa il riferimento immutabile e decentralizzato del lotto, ed è registrato nello smart contract come riferimento del token ERC-1155 corrispondente.
- **Blockchain Ethereum (Ganache)**: custodisce le operazioni sui token, ossia le primitive di creazione (mint), trasferimento (transfer), suddivisione (split/merge) e

distruzione (burn). Ogni lotto è rappresentato da un token ERC-1155, il cui URI punta al contenuto IPFS. Le transazioni e gli eventi generati dal contratto costituiscono la prova immutabile delle trasformazioni del lotto.

- **MongoDB**: agisce come livello ausiliario e aggregatore. Integra i riferimenti ai dati IPFS, le evidenze on-chain (transazioni ed eventi) e le informazioni provenienti dalle richieste HTTP. In questo modo arricchisce la prospettiva tecnica con i dati di business delle operazioni, permettendo interrogazioni efficienti e visualizzazioni mirate senza sostituire le evidenze primarie garantite da blockchain e IPFS.

Questa articolazione a tre livelli garantisce equilibrio tra **immutabilità** (blockchain), **descrittività e contenuto** (IPFS) ed **efficienza di consultazione** (MongoDB). In tal modo, ogni lotto risulta caratterizzato da un'identità univoca, da una scheda informativa immutabile e da una cronologia operativa interrogabile. Descritta l'architettura del sistema diventa possibile concentrarsi sul cuore del progetto, lo smart contract, che definisce quali informazioni sono registrate sulla blockchain e quali operazioni possono essere eseguite sui lotti. Questo passaggio prepara l'analisi del capitolo successivo, in cui verranno presentati gli endpoint sviluppati e le modalità con cui l'applicazione li rende fruibili.

Smart contract

Il cuore del sistema è rappresentato dallo smart contract ***SupplyChain1155*** *(Fig. 16)* sviluppato in Solidity e costruito a partire dallo standard ERC-1155. Il contratto eredita le interfacce ERC1155, ERC1155Burnable e Ownable: la prima offre le primitive di ERC1155, la seconda aggiunge la possibilità di bruciare token in maniera controllata, mentre la terza introduce un meccanismo di access control che limita a ruoli autorizzati l'esecuzione di specifiche operazioni. Rispetto alla specifica originaria di ERC-1155, SupplyChain1155 è stato esteso in più punti per rispondere alle necessità di un dominio in cui i lotti non sono semplici token indistinti ma entità digitali portatrici di metadati complessi. La prima modifica riguarda la gestione degli URI: mentre lo standard prevede un URI base al quale concatenare l'identificativo del token, in SupplyChain1155 è stata introdotta una mappa diretta da tokenId a URI. Questo consente di associare a ciascun lotto un URI completo e unico, che corrisponde al CID IPFS derivato dall'hash del documento caricato in rete. Tale scelta garantisce l'immutabilità e la verificabilità dei contenuti: modificare anche di un singolo bit il documento genererebbe un CID diverso, rendendo immediatamente rilevabili manomissioni o sostituzioni.

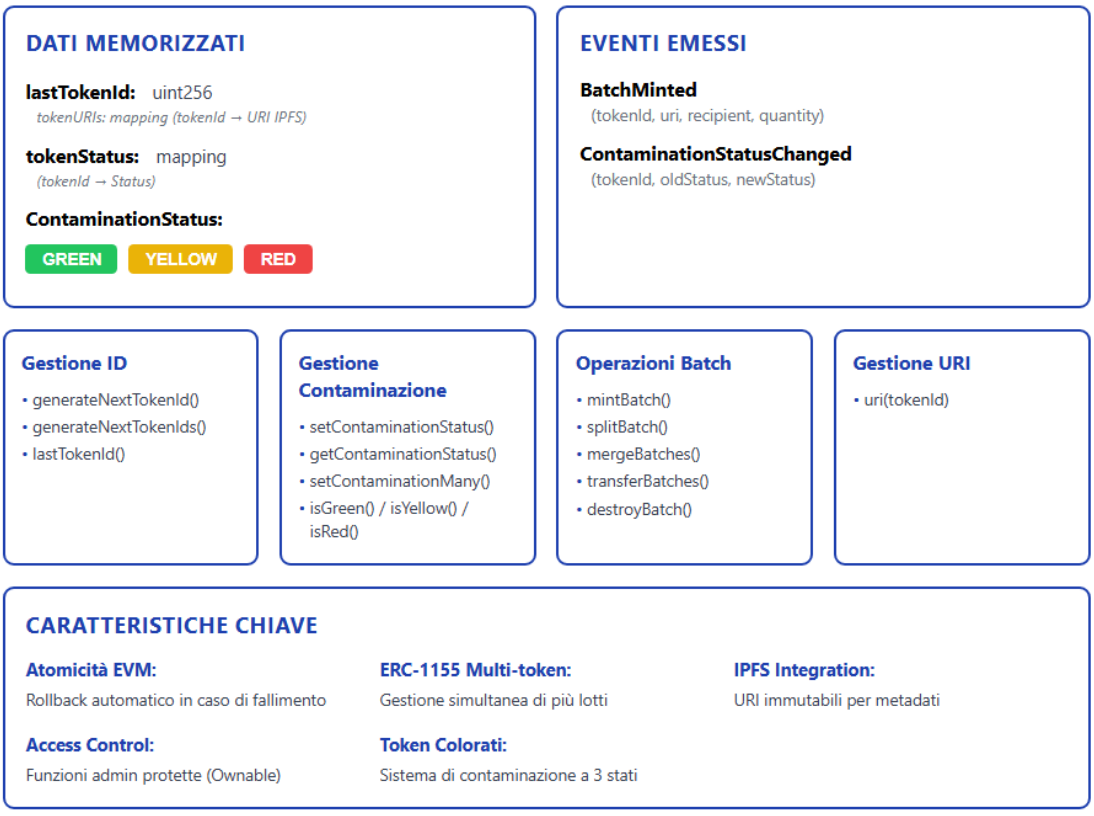


Figura 16: Visione Semplificata dello Smart Contract SupplyChain1155

Un secondo elemento chiave è il sistema di generazione degli identificativi. Invece di affidare la creazione degli ID al client, con il rischio di collisioni o incoerenze, il contratto fornisce le funzioni generateNextId e generateNextIds, che producono identificativi sequenziali on-chain. La gestione è quindi delegata alla blockchain stessa: l'Ethereum Virtual Machine (EVM) esegue ogni transazione in isolamento e gestisce eventuali conflitti attraverso il proprio meccanismo di validazione e consenso. In Ganache, per semplificare lo sviluppo, è stato abilitato l'automine, così che ogni transazione produce immediatamente un blocco e rende deterministico l'ordine delle operazioni. Lo smart contract implementa tutte le operazioni fondamentali del ciclo di vita dei lotti. La creazione (mintNewBatch) utilizza la primitiva _mint di OpenZeppelin ed è accompagnata dall'emissione di un evento BatchMinted. Il trasferimento (transferBatches) incapsula la primitiva safeBatchTransferFrom per spostare più lotti in una singola transazione. La suddivisione (splitBatch) è modellata come un'operazione composta: il batch originario viene prima bruciato tramite _burn e quindi rimpiazzato da più nuovi batch generati con _mint. La fusione (mergeBatches) segue la logica opposta: più lotti vengono bruciati e rimpiazzati da un unico lotto di output. Per la trasformazione è stato scelto di riutilizzare proprio la logica di merge: a differenza di modelli più complessi che prevedono trasformazioni N a M, qui è stata implementata una trasformazione N a 1. Tale scelta riduce la complessità implementativa e sfrutta le stesse primitive già testate nel merge, semplificando il modello. La distruzione è gestita attraverso destroyBatch, che invoca _burn.

Un'estensione *(Fig. 17)* è stata introdotta per testare la propagazione di stati di contaminazione. A ciascun token può essere associato uno stato GREEN, YELLOW o RED. Le regole di propagazione stabiliscono che un token contaminato implichi contaminazione diretta nei lotti figli da questo generati e potenziale contaminazione nei lotti genitori. Lo stato condiziona le operazioni: i token verdi sono pienamente operativi, quelli gialli possono essere riportati a verde o distrutti, mentre i rossi possono solo essere bruciati. Pur non facendo parte del nucleo della tracciabilità, questa estensione illustra la possibilità di inserire regole di validazione addizionali a supporto di scenari più avanzati.

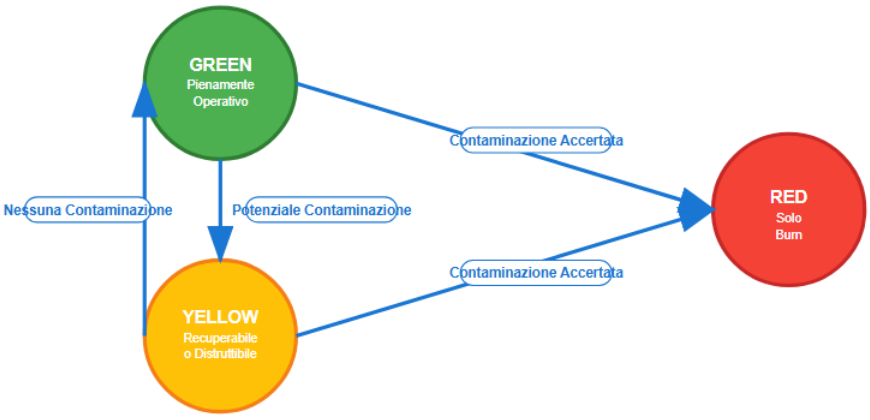


Figura 17: Diagramma di Stato di Contaminazione

Dal punto di vista della robustezza, le funzioni del contratto ereditano atomicità dall'EVM. Ogni operazione complessa, come ad esempio uno split che comporta un _burn e più _mint, è trattata come un'unica transazione. L'EVM gestisce questa proprietà lavorando su una copia temporanea dello stato globale: se l'esecuzione termina con successo, le modifiche vengono applicate, se invece si verifica un errore o un revert, la copia viene scartata e lo stato rimane invariato. In questo modo si evita che operazioni parziali lascino il sistema in condizioni incoerenti.

L'architettura descritta costituisce lo strato interno del sistema, mentre lo smart contract ne rappresenta il nucleo computazionale. Nel capitolo successivo verranno analizzati i flussi operativi esposti tramite API REST.

Capitolo 6: Implementazione

In questo capitolo passiamo dalla descrizione architetturale ai **flussi di utilizzo** esposti tramite API. Presentiamo prima una **vista d'insieme** degli endpoint, poi entriamo nel dettaglio con esempi e schede operative. La trattazione è focalizzata sui **lotti (batches)**, suddivisi in **collection level** (operazioni su insiemi di lotti, con riferimenti specificati nel payload) e **resource level** (operazioni su un singolo lotto identificato nel path). Questa impostazione consente di leggere gli endpoint come **strumenti di processo**.

Endpoint esposti: struttura generale ed esempi

Gli endpoint costituiscono l'interfaccia verso il sistema di tracciabilità e sono progettati secondo principi REST: URI semantici, metodi HTTP con significato chiaro, payload in JSON (o multipart/form-data quando sono presenti allegati) e risposte uniformi con dati ed errori coerenti. La trattazione si concentra sugli endpoint dei lotti (batches), direttamente connessi agli obiettivi della tesi. Gli altri endpoint, come attori e prodotti, aprono a considerazioni architetturali imprescindibili ma non centrali rispetto al focus, in particolare la necessità di un sistema di gestione dell'identità e delle chiavi degli utenti. Questi aspetti saranno accennati nei capitoli finali come prospettive di evoluzione. La tabella seguente *(Fig. 18)* sintetizza gli endpoint **collection level** relativi ai batches e le loro principali interazioni con i servizi; nelle pagine successive completeremo la panoramica con gli endpoint **resource level** e le schede di dettaglio. Gli endpoint di livello **collection** operano su insiemi di lotti e richiedono di specificare nel **payload** a quali batch si fa riferimento. Sono ideali per operazioni multi-lotto come **merge**, **transform** e **transfer**, dove l'input comprende una lista di batchId e quantità. In questa modalità la selezione dei lotti avviene nel corpo della richiesta e l'API orchestra in un'unica transazione applicativa tutte le verifiche di coerenza. Gli endpoint di livello **resource** *(Fig. 19)* lavorano invece su un **singolo lotto** identificato nel **path** (/batches/:batchId).

Endpoint	Descrizione e Interazioni
`POST /batches`	**Crea un nuovo batch** (dati pubblici + eventuali allegati). *Interazioni: IPFS (allegati/metadata), Blockchain (mint), MongoDB (upsert).*
`GET /batches`	**Elenca i batch** (liste, filtri, paginazione). *Interazioni: MongoDB (query).*
`POST /batches/merge`	**Unisce N batch in 1** (lotto di output). *Interazioni: Blockchain (burn+mint), IPFS (nuovi metadata), MongoDB (aggiornamenti).*
`POST /batches/transform`	**Trasforma N→1** con **nuovo prodotto/metadata**. *Interazioni: Blockchain, IPFS, MongoDB.*
`POST /batches/transfer`	**Trasferisce più batch** in un'unica richiesta. *Interazioni: Blockchain (safeBatchTransfer), MongoDB (operation log).*
`POST /batches/contaminate`	**Imposta e propaga** lo stato di contaminazione **su più batch** (propagazione a valle sull'albero di derivazione). *Interazioni: Blockchain (stato), MongoDB (propagazione/stato).*
`GET /batches/validate`	**Verifica la consistenza** (legami, quantità, storia). *Interazioni: MongoDB (ricostruzione grafi) + eventi Blockchain (cross-check).*

Figura 18: Alcuni Endpoint per la gestione di lotti

Endpoint	Descrizione e Interazioni
`GET /batches/:batchId`	**Dettaglio** di un batch. *Interazioni: MongoDB (documento + riferimenti IPFS).*
`GET /batches/:batchId/parents`	**Genitori** (provenienza). *Interazioni: MongoDB (grafi/operazioni).*
`GET /batches/:batchId/descendants`	**Discendenti** (derivazioni). *Interazioni: MongoDB (grafi/operazioni).*
`GET /batches/:batchId/trace-history`	**Cronologia di tracciabilità**. *Interazioni: MongoDB (+ eventi on-chain per verifica).*
`POST /batches/:batchId/split`	**Divide 1 in N** lotti figli. *Interazioni: Blockchain (burn+mint), IPFS (metadata figli), MongoDB.*
`POST /batches/:batchId/transfer`	**Trasferisce** un singolo batch. *Interazioni: Blockchain (safeTransfer), MongoDB (operation log).*
`DELETE /batches/:batchId`	**Destroy/Burn** (totale o parziale). *Interazioni: Blockchain (burn), MongoDB (quantità/stato).*
`GET /batches/:batchId/status`	**Stato** (contaminazione, ciclo di vita). *Interazioni: MongoDB (vista stato).*
`PUT /batches/:batchId/contamination`	**Imposta lo stato di contaminazione** del batch (operazione puntuale). *Interazioni: Blockchain (stato), MongoDB (aggiornamento).*

Figura 19: Alcuni Endpoint per la gestione di un lotto

Ciascun endpoint dell'API restituisce una HTTP response con un codice di stato che descrive l'esito della richiesta. In Fig. 20, 21 sono riportati alcuni dei principali codici di errore che l'API può restituire, accompagnati da una descrizione sintetica e dalle cause tipiche che li generano, così da rendere prevedibile la gestione degli esiti lato client.

Codice	Descrizione
200	**OK** Lettura e operazioni condotte correttamente. `{ "success": true, "data": { ... } }`
201	**Created** Creazione di una nuova risorsa completata con successo. `{ "success": true, "data": { "id": "123", "created": "2024-01-01T10:00:00Z" } }`
204	**No Content** Operazione riuscita senza corpo di risposta (tipicamente per DELETE). *Nessun body nella risposta*
400	**Bad Request** Payload mancante o malformato, parametri non validi, violazione delle regole di business. `{ "success": false, "error": "validation failed", "code": 400, "details": "field 'quantity' is required" }` **Cause tipiche:** • JSON malformato • Campi obbligatori mancanti • Valori fuori range • Formato dati non corretto
403	**Forbidden** Utente autenticato ma non autorizzato per questa operazione. `{ "success": false, "error": "forbidden", "code": 403, "details": "insufficient permissions or ownership required" }` **Cause tipiche:** • Operazione su risorsa non posseduta • Ruolo insufficiente • Violazione politiche di accesso
404	**Not Found** Risorsa richiesta non esistente o non accessibile. `{ "success": false, "error": "not found", "code": 404, "details": "resource with id '123' does not exist" }` **Cause tipiche:** • ID risorsa inesistente • Endpoint non implementato • Risorsa eliminata

Figura 20: Alcuni codici errore restituiti dall'API

Codice	Descrizione
409	**Conflict** Conflitto di stato della risorsa o violazione di vincoli di dominio. `{ "success": false, "error": "conflict", "code": 409, "details": "cannot merge batches: quantity mismatch" }` **Cause tipiche:** • Stato risorsa incompatibile • Violazione regole business • Duplicazione chiavi uniche
422	**Unprocessable Entity** Dati sintatticamente corretti ma semanticamente non validi. `{ "success": false, "error": "unprocessable entity", "code": 422, "details": "business rule validation failed" }` **Cause tipiche:** • Validazioni di dominio fallite • Regole business non rispettate • Dati inconsistenti
429	**Too Many Requests** Rate limiting attivo - troppe richieste in un intervallo di tempo. `{ "success": false, "error": "too many requests", "code": 429, "retryAfter": 60 }` **Gestione:** • Implementare backoff esponenziale • Rispettare header Retry-After • Ridurre frequenza richieste
500	**Internal Server Error** Errore generico del server non previsto o gestito. `{ "success": false, "error": "internal server error", "code": 500, "requestId": "req_123456" }` **Cause tipiche:** • Bug nel codice server • Errori di configurazione • Eccezioni non gestite • Problemi infrastruttura
502	**Bad Gateway** Errore da servizi esterni o dipendenze (blockchain, IPFS, database). `{ "success": false, "error": "bad gateway", "code": 502, "details": "blockchain service timeout" }` **Servizi interessati:** • Blockchain networks • IPFS storage • Database esterni • API di terze parti
503	**Service Unavailable** Servizio temporaneamente non disponibile per manutenzione o sovraccarico. `{ "success": false, "error": "service unavailable", "code": 503, "retryAfter": 300 }` **Gestione:** • Retry automatico dopo intervallo • Implementare circuit breaker • Monitorare stato servizio

Figura 21: Alcuni codici errore restituiti dall'API (II)

Implementazione degli endpoint e diagrammi UML di sequenza

Per analizzare i flussi operativi, è utile partire dal primo e più importante endpoint del sistema, ovvero la creazione di un batch. Questo endpoint rappresenta il punto di ingresso di qualsiasi processo di tracciabilità: consente di registrare in modo immutabile un nuovo lotto di prodotto, associandogli dati pubblici, metadati descrittivi e allegati documentali *(Fig.22)*.

Create:

La chiamata a POST /supply-chain/batches innesca un flusso che coinvolge **tutti i livelli architetturali** descritti nel capitolo precedente:

- validazione lato API,
- archiviazione allegati e metadati su **IPFS**,
- registrazione dell'identità del batch sulla **blockchain Ethereum (ERC-1155)**,
- salvataggio dei dati operativi su **MongoDB**.

Un **caso di studio ideale** per comprendere la logica del sistema presentato con una descrizione strutturata, seguita da un **diagramma di sequenza** che mostra le interazioni tra client, API e servizi esterni.

Sezione	Contenuto
Endpoint	POST /supply-chain/batches Content-Type: multipart/form-data
Descrizione	Crea un nuovo batch ERC-1155 con dati pubblici, allegati documentali, metadati IPFS e registrazione immutabile su blockchain.
Request (FormData)	- json: stringa JSON con dati batch - attachments: array di file allegati
Request (JSON payload)	`json { "batchCreator": {...}, "product": {...}, "quantity": 1000, "unitOfMeasure": "kg", "operation": {...}, "attachments_metadata": [...] }`
Response (201 Created)	`json { "success": true, "data": { "batchId": "123", "publicData": {...}, "traceabilityData": {...} }, "meta": { "gas": {...} } }`
Flusso operativo	1. Validazione input e allegati 2. Generazione ID batch 3. Upload allegati su IPFS 4. Creazione metadati batch 5. Mint su blockchain 6. Salvataggio in MongoDB 7. Response al client
Errori tipici	• 400: JSON mancante o invalido • 403: Indirizzo non autorizzato • 409: ID duplicato • 500: errore IPFS o blockchain

Figura 22: Creazione di un batch

La *Fig. 22* riassume l'endpoint di creazione lotto (POST /supply-chain/batches) in formato tabellare. All'inizio sono indicati metodo, URL e content-type (multipart/form-data), seguiti da una descrizione dell'obiettivo: creare un nuovo batch ERC-1155, associare dati pubblici e allegati, produrre i metadati su IPFS e registrare l'operazione su blockchain. La sezione Request (FormData) è composta da due parti: il campo json con il payload strutturato del lotto (attore, prodotto, quantità, unità di misura, operazione, metadati degli allegati) e l'array attachments con i file. La sezione Response (201 Created) utilizza l'envelope standard con success: true e include almeno batchId, publicData e traceabilityData con URI IPFS ed evidenze on-chain. Il riquadro Flusso operativo elenca i passaggi principali: validazione, generazione dell'ID, upload su IPFS di allegati e metadati, operazione di mint su blockchain, persistenza su MongoDB e risposta al client. La sezione Errori tipici riporta i codici più frequenti con le relative cause: 400 per input non valido, 403 per assenza di autorizzazione, 409 per conflitti di stato, 502 e 503 per problemi con servizi esterni come IPFS o blockchain. A seguire *(Fig. 23)* è riportato il diagramma di sequenza, che dettaglia l'ordine delle interazioni tra client, API, IPFS, blockchain e livello dati.

Quando l'API layer riceve la richiesta, esegue parsing e validazione, quindi avvia il processo di creazione. Per prima cosa vengono caricati gli allegati su IPFS così da ottenere gli URI immutabili dei file. Ricevuti questi URI, si costruisce la scheda metadati del lotto combinando il corpo richiesta e i riferimenti IPFS degli allegati; la scheda viene poi pubblicata su IPFS. Il risultato è un documento IPFS del batch (logicamente associato al batch_id) che contiene i dati di creazione e i link immutabili ai suoi allegati. Questa pubblicazione genera un nuovo URI IPFS che diventa il riferimento da usare per la creazione del token on-chain. A questo punto si invoca la transazione di mint del token ERC-1155 utilizzando l'URI IPFS del batch. Se la transazione va a buon fine, la blockchain restituisce una ricevuta con parametri di successo (hash di transazione, numero di blocco, gas utilizzato, eventuali eventi). Queste informazioni vengono aggregate con i dati della richiesta iniziale e con i riferimenti IPFS e quindi persistite in MongoDB. Solo dopo il salvataggio l'API restituisce al client la response finale con stato 201 Created e i dati del lotto.

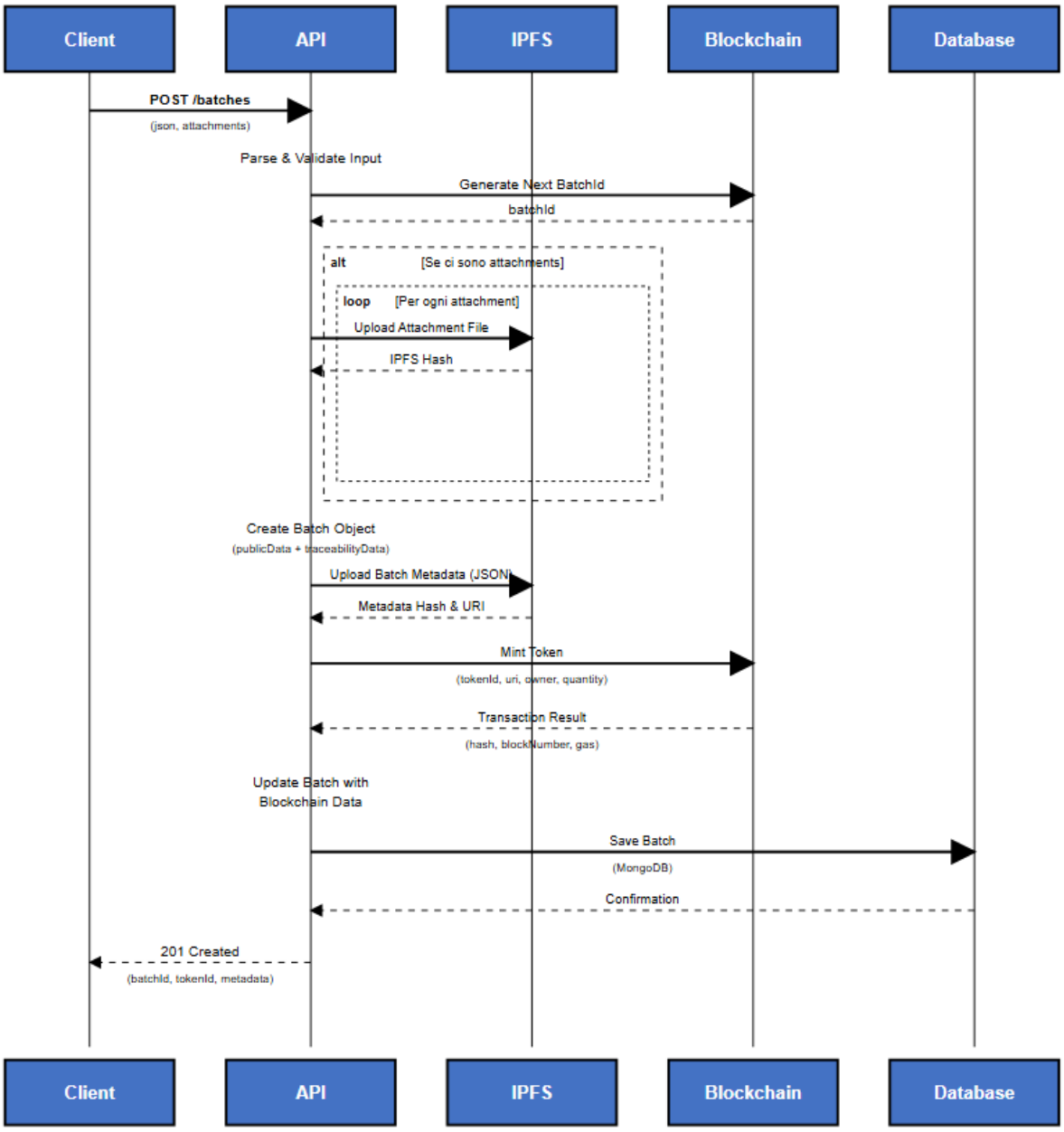


Figura 23: Diagramma di sequenza per la creazione

Transfer:

Dopo la trattazione dell'endpoint di creazione, analizziamo l'endpoint di trasferimento, che, dal punto di vista semantico, realizza l'atto unitario di cessione e acquisizione della titolarità su un lotto o parte di esso tra due attori. L'operazione è di tipo resource level, agisce su un batch esistente identificato nel path, e registra on chain il passaggio di proprietà, preservando continuità e verificabilità della traccia storica. Le principali precondizioni riguardano la validità degli indirizzi, la disponibilità della quantità da trasferire, la dimostrabilità dell'ownership da parte del mittente e la trasferibilità dello stato del lotto. La Figura 24 presenta la scheda riassuntiva dell'endpoint, mentre la Figura 26 illustra il corrispondente diagramma di sequenza, evidenziando la catena di controlli e l'aggiornamento coerente degli artefatti applicativi e on chain.

Sezione	Contenuto
Endpoint	POST /supply-chain/batches/{batchId}/transfer • Content-Type: application/json
Descrizione	Trasferisce un lotto esistente da un indirizzo a un nuovo destinatario, registrando l'operazione on-chain e aggiornando la tracciabilità applicativa.
Request (FormData)	Non applicabile (la richiesta usa JSON).
Request (JSON payload)	`json { "owner": "0x742d35CCE634C0532925a3b8D4C9db964b4d8b6", "to": "0x8ba1f109551bD432883012645fac136c4c8b8bb8", "quantity": 500 }`
Response (200 OK)	`json { "success": true, "data": { "transferredBatch": { "batchId": "123", "from": "0x742d35...", "to": "0x8ba1f1...", "quantity": 500, "operationId": "op-170531220000B-transfer", "txhash": "0xdef45e...", "blockNumber": 12346 } } }`
Flusso operativo	1. Validazione input 2. Verifica esistenza batch 3. Controllo ownership e balance on-chain 4. Verifica stati del batch (es. non contaminato, quantità disponibile) 5. Esecuzione transfer su blockchain 6. Aggiornamento tracciabilità in MongoDB 7. Response al client
Errori tipici	• 400: input non valido o saldo insufficiente • 403: permessi o ownership insufficienti • 404: batch non trovato • 500: errore blockchain o persistenza

Figura 24: Trasferimento di un batch

Si noti la presenza del campo owner, che identifica l'indirizzo del wallet a nome del quale viene effettuata la cessione e l'acquisizione del lotto. Nel prototipo questo campo è richiesto perché, in fase di sviluppo, il backend firma le transazioni utilizzando chiavi e indirizzi configurati nel file .env del progetto (*Fig. 25*), operando quindi "per conto" dell'utente indicato nella richiesta. **Tale soluzione è funzionale a scopi dimostrativi e di test, ma non è adeguata per un ambiente di produzione. In uno scenario reale è necessario introdurre un sottosistema di autenticazione e gestione delle chiavi che consenta, alternativamente, la firma diretta da parte dell'utente** con le proprie credenziali (self-custody) oppure l'utilizzo di un servizio centralizzato che custodisce le chiavi e firma secondo policy controllate (custody). Questi aspetti, pur non rientrando nel perimetro della presente trattazione, sono ripresi nella sezione conclusiva dedicata a estensioni, criticità e sviluppi futuri, assumendo che in produzione gli utenti siano autenticati e che le firme avvengano in modo conforme al modello di sicurezza prescelto.

```
# Admin (usato per il deploy)
ADMIN_ADDRESS=0x666d7caB9F275446d1Ae88cD9baEB9687D73F235
ADMIN_PRIVATE_KEY=0x2118335360329fce4b2acc5bf72dfd48b30e47349e007c8a214319f4aaf5bc1

# Account 1
ACCOUNT1_ADDRESS=0x141985dE74608aa92f0DE0a7F841004046B3D09F
ACCOUNT1_PRIVATE_KEY=0x64e3e9a36708b9289c4e6abb391040eb5e094e0bf5b5eb36529a229e2a8435ba

# Account 2
ACCOUNT2_ADDRESS=0x555C263c0eec8D02503a984b61347D4373F365d4
ACCOUNT2_PRIVATE_KEY=0x71ba4cd6d3d0a1e3e207c4e30e88bd258f9aa9f0cf1773aad9908eeec00fe6c9

# Account 3
ACCOUNT3_ADDRESS=0xFd8E1d22B3d07Fe33b803346a728693B2C26ab50
ACCOUNT3_PRIVATE_KEY=0x3f775f5f33cf999487eeb9f265962da2f66031ed1af7146377a621619a8de757
```

Figura 25: Estratto .env, credenziali simulate su Ganache

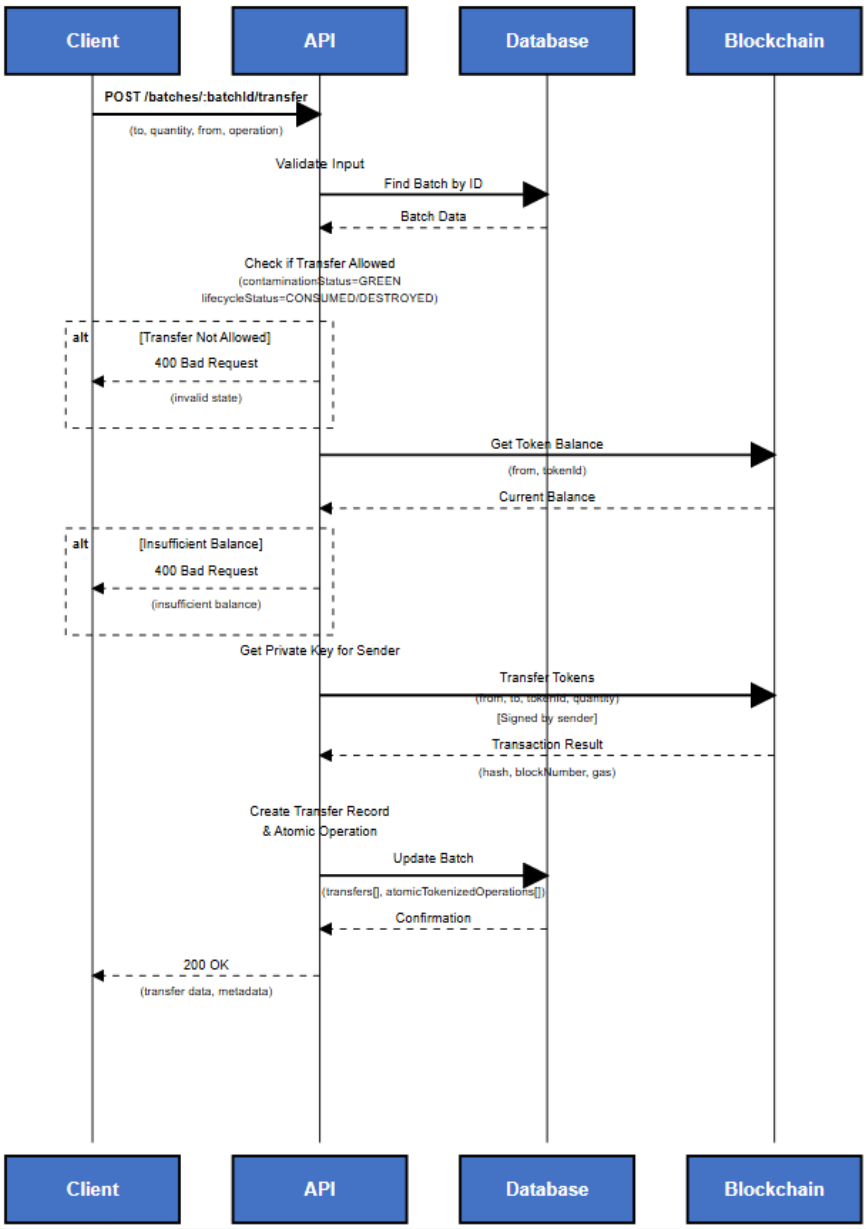


Figura 26: Diagramma di sequenza per il trasferimento

La *Fig. 26* illustra, in forma di sequenza, il flusso dell'endpoint di trasferimento: il client invia una richiesta HTTP POST a /supply-chain/batches/{batchId}/transfer con corpo JSON composto da owner (indirizzo del mittente a nome del quale si intende operare), to (indirizzo del destinatario) e quantity (quantità da trasferire). All'arrivo della richiesta, l'API esegue parsing e un insieme di validazioni che comprende il controllo del formato del payload e degli indirizzi, la verifica dell'esistenza del batch e dell'integrità dei suoi dati, l'accertamento dell'ownership e del saldo on-chain per il token indicato, nonché la trasferibilità rispetto agli stati applicativi. Superate le verifiche, il servizio blockchain esegue la transazione di trasferimento sullo smart contract ERC-1155 e ne attende la conferma; l'esito della transazione (hash, numero di blocco, eventuali eventi) viene quindi restituito al livello applicativo, che aggiorna la tracciabilità del batch nel database aggiungendo il record di transfer e, ove necessario, adeguando gli attributi derivati. Il flusso si conclude con la risposta al client 200 OK contenente i dati essenziali dell'operazione e i riferimenti all'esecuzione on chain.

Split:

Il terzo endpoint fondamentale è lo *split* di un batch, ossia la possibilità di suddividere un lotto esistente in più lotti figli, ciascuno con quantità e metadati specifici. Dal punto di vista semantico, lo split rappresenta un'operazione di trasformazione interna: il batch originale viene "consumato" (burn) e al suo posto nascono nuovi token ERC-1155 che ereditano la tracciabilità del genitore. La *Fig. 27* riporta la scheda riassuntiva dell'endpoint di split, mentre la Figura 28 illustrerà il diagramma di sequenza che mostra la catena di controlli, l'aggiornamento degli artefatti applicativi e l'esecuzione on-chain.

Sezione	**Contenuto semplificato**
Endpoint	POST /supply-chain/batches/{batchId}/split
Descrizione	Divide un batch in più batch figli, bruciando l'originale e creando nuovi token.
Request (JSON)	`{ "owner": "0x742d...", "splits": [ { "quantity": 300, "unitOfMeasure": "kg" }, { "quantity": 200, "unitOfMeasure": "kg" } ] }`
Response (200)	`{ "success": true, "burnedBatchId": "123", "newBatches": [ { "batchId": "456", "quantity": 300 }, { "batchId": "457", "quantity": 200 } ] }`
Flusso operativo	1. Validazione input 2. Verifica batch esistente e stato valido 3. Controllo ownership e quantità 4. Generazione ID token 5. Upload metadati su IPFS (se allegati) 6. Split on-chain (burn + mint) 7. Aggiornamento batch originale come consumato 8. Creazione nuovi batch in DB 9. Invio response
Errori tipici	400 input non valido o quantità errate 403 permessi insufficienti 404 batch non trovato 500 errore blockchain/IPFS/DB

Figura 27: Split di un batch

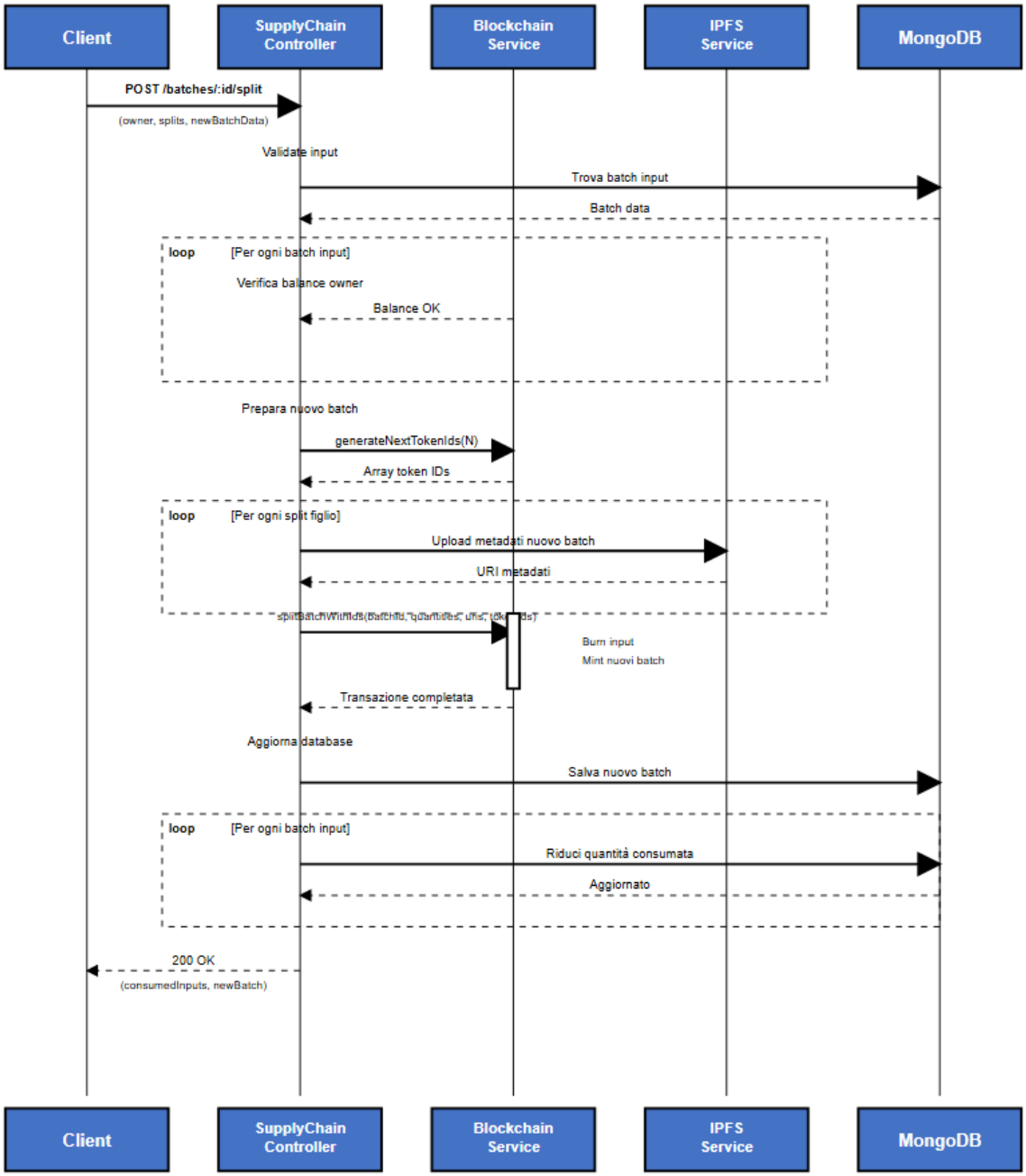


Figura 28: Diagramma di sequenza per lo split

La Fig. 28 illustra, in forma di sequenza, il flusso dell'endpoint di split: il client invia una richiesta HTTP POST /supply-chain/batches/{batchId}/split con corpo JSON contenente l'indirizzo del proprietario (owner) e l'array delle suddivisioni (splits) dei nuovi lotti da generare. All'arrivo della richiesta, l'API Layer effettua il parsing del payload e avvia le procedure di validazione, verificando l'esistenza del batch originale, il suo stato applicativo (deve essere attivo e non consumato o contaminato) e la disponibilità della quantità indicata. Superata questa fase, i servizi applicativi recuperano i dati del batch, controllano il saldo on-chain del token corrispondente e si assicurano che il proprietario disponga di una quantità sufficiente per eseguire lo split. In caso positivo, viene invocata l'operazione di split sullo smart contract ERC-1155, che in maniera atomica brucia il lotto originale e crea i nuovi token figli con i rispettivi URI IPFS dei metadati. La blockchain restituisce quindi l'esito della

transazione (hash, numero di blocco ed eventi generati), che viene propagato ai servizi applicativi. Il livello servizi aggiorna di conseguenza lo stato del database: il batch padre viene marcato come consumato, mentre i batch figli vengono creati con i relativi attributi iniziali. Completato l'aggiornamento, l'API Layer costruisce la risposta finale al client con stato 200 OK, includendo i dati dei nuovi batch e i riferimenti all'esecuzione on-chain.

Merge:

Il quarto endpoint è quello di *merge*, che completa il ciclo operativo dopo la creazione, il trasferimento e lo split. L'operazione consente di combinare più batch esistenti in un unico nuovo lotto, utile per scenari di blending. Dal punto di vista semantico rappresenta un atto di **composizione**: i batch input vengono consumati (parzialmente o completamente) e al loro posto viene creato un nuovo token ERC-1155 che eredita la tracciabilità dei genitori. La Figura 29 riassume la struttura dell'endpoint POST /supply-chain/batches/merge. La request prevede l'indirizzo del proprietario (owner), l'elenco dei batch in ingresso (inputs con batchId e quantity per ciascun lotto) e la definizione del nuovo lotto (newBatchData) con quantità, unità di misura, prodotto e metadati. La response restituisce l'elenco dei batch consumati, i dati del nuovo batch (id, quantità, stato iniziale, relazioni padre-figlio), oltre ai riferimenti tecnici della transazione on-chain (hash, block number) e agli URI IPFS. Il flusso operativo elencato nella tabella mette in evidenza i passaggi principali: validazione, controlli su stato e disponibilità, upload dei metadati su IPFS, esecuzione atomica del burn/mint su blockchain, aggiornamento dei batch in ingresso e creazione del nuovo batch nel database.

Sezione	**Contenuto semplificato**
Endpoint	POST /supply-chain/batches/merge
Descrizione	Combina più lotti esistenti in un unico nuovo lotto, bruciando le quantità specificate dai lotti input e creando un nuovo token con metadati consolidati.
Request (JSON)	`{ "owner": "0x742d35CCE634C0532925a3b8D4C9db964b4d8b6", "inputs": [{ "batchId": "123", "quantity": 300 }, { "batchId": "124", "quantity": 200 }], "newBatchData": { "quantity": 500, "unitOfMeasure": "kg", "product": { "name": "Mixed Grain Blend", "category": "Processed" }, "operation": { "operationName": "Quality Merge" } } }`
Response (200)	`{ "success": true, "data": { "consumedInputs": [{ "batchId": "123", "quantity": 300 }, { "batchId": "124", "quantity": 200 }], "newBatch": { "batchId": "789", "quantity": 500, "contaminationStatus": "GREEN", "lifecycleStatus": "UNUSED", "parentBatchesIds": ["123", "124"] } }, "meta": { "txhash": "0xdef45e...", "newTokenId": "789" } }`
Flusso operativo	1. Validazione input (owner, inputs array, newBatchData) 2. Verifica esistenza batch input e stati (solo GREEN non consumato/distrutto) 3. Controllo ownership e balance on-chain per ogni input batch 4. Upload metadati IPFS per il nuovo lotto con allegati 5. Esecuzione merge on-chain (burn input + mint nuovo batch) 6. Creazione nuovo batch con relazioni padre-figlio 7. Aggiornamento input batches 8. Persistenza nuovo batch e aggiornamenti in MongoDB 9. Response al client con dettagli operazione
Errori tipici	400 input non valido, saldo insufficiente, batch non trovati 403 permessi insufficienti o batch non in stato valido per merge 404 uno o più batch input non trovati 500 errore blockchain, IPFS o persistenza 400 batch contaminati (RED) o completamente consumati/distrutti

Figura 29:Merge di un batch

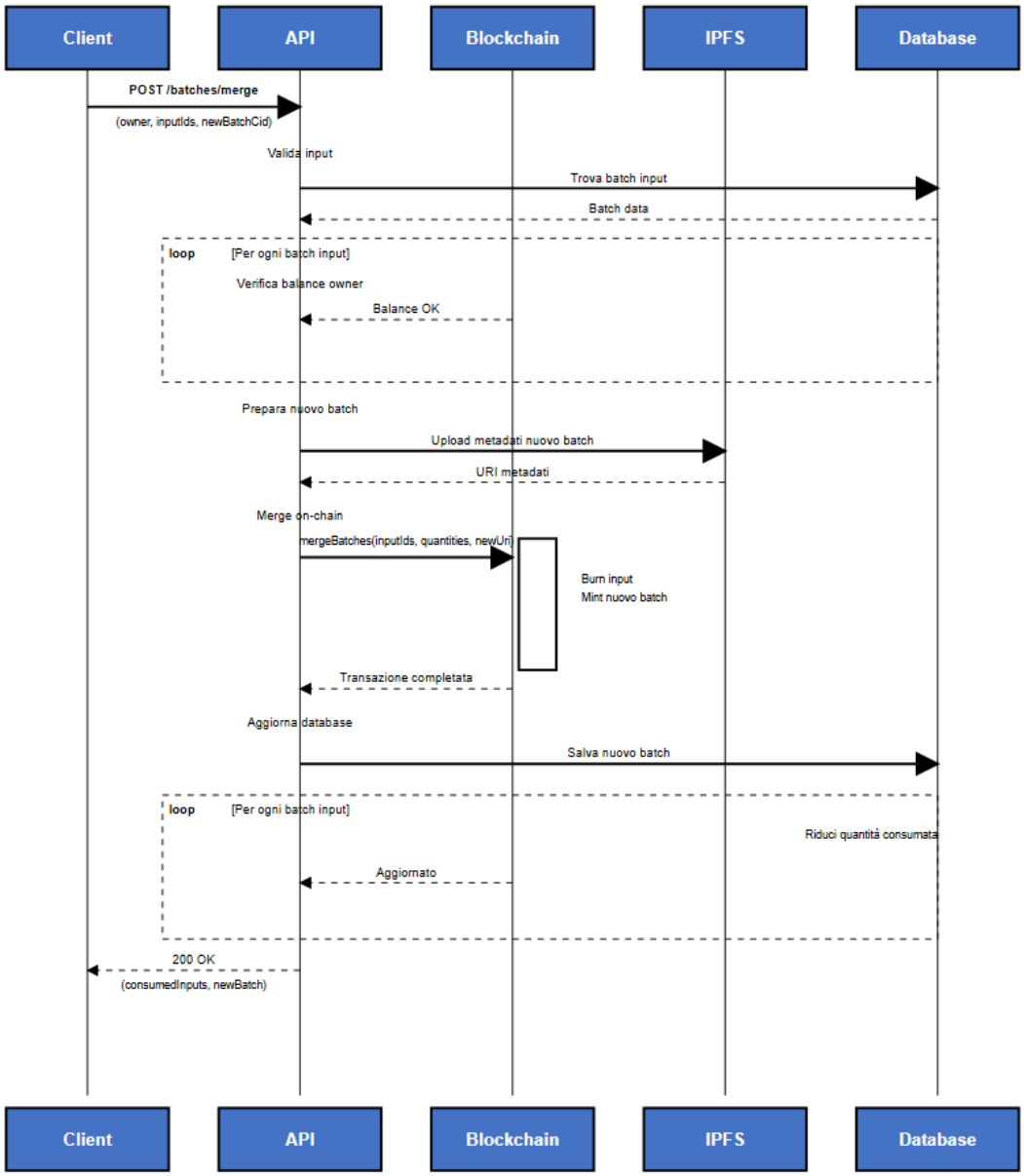


Figura 30: Diagramma di sequenza per il Merge

La Fig. 30 illustra, in forma di sequenza, il flusso dell'endpoint di **merge**: il client invia una richiesta HTTP POST /supply-chain/batches/merge con corpo JSON contenente l'indirizzo del proprietario (owner), l'elenco dei batch in ingresso (inputs) con le quantità da fondere e la definizione del nuovo lotto (newBatchData). All'arrivo della richiesta, l'API Layer effettua il parsing del payload e avvia le procedure di validazione, verificando l'esistenza dei batch originali, il loro stato applicativo (devono essere attivi e non contaminati o distrutti) e la disponibilità delle quantità dichiarate. Superata questa fase, i servizi applicativi recuperano i dati dei batch, controllano i saldi on-chain dei token corrispondenti e si assicurano che il proprietario disponga delle quantità necessarie per eseguire il merge. In caso positivo, viene invocata l'operazione di merge sullo smart contract ERC-1155, che in maniera atomica brucia le quantità consumate dei batch di ingresso e crea un nuovo token figlio con URI

IPFS consolidato. La blockchain restituisce quindi l'esito della transazione (hash, numero di blocco ed eventi generati), che viene propagato ai servizi applicativi.

Il livello servizi aggiorna di conseguenza lo stato del database: i batch genitori vengono marcati come consumati o parzialmente consumati, mentre il nuovo batch viene creato con i relativi attributi iniziali. Completato l'aggiornamento, l'API Layer costruisce la risposta finale al client con stato 200 OK, includendo i dati del nuovo batch e i riferimenti all'esecuzione on-chain.

Transform:

Il quinto endpoint è quello di *transform*, l'operazione consente di **trasformare uno o più batch esistenti in un nuovo lotto con caratteristiche diverse**, ad esempio quando una materia prima viene lavorata in un prodotto finito o semilavorato. Dal punto di vista semantico, il transform non si limita a una mera combinazione ma rappresenta un vero e proprio **cambiamento di natura** del prodotto: i batch in ingresso vengono consumati (totalmente o parzialmente) e al loro posto viene generato un nuovo token ERC-1155 con metadati aggiornati (nome, categoria, descrizione), allegati specifici e riferimenti alla lavorazione. La Figura 31 riassume la struttura dell'endpoint POST /supply-chain/batches/transform. La request prevede l'indirizzo del proprietario (owner), l'elenco dei batch input (inputs con batchId e quantity) e la definizione del nuovo lotto (newBatchData) con quantità, unità di misura, dati prodotto aggiornati e allegati. La response restituisce i batch consumati, il nuovo batchId con i suoi attributi iniziali e i riferimenti tecnici all'operazione on-chain (hash, block number) e ai metadati IPFS.

Sezione	**Contenuto semplificato**
Endpoint	POST /supply-chain/batches/transform
Descrizione	Trasforma uno o più lotti esistenti in un nuovo lotto con caratteristiche diverse (es. processo di lavorazione, cambio prodotto, trasformazione fisica/chimica).
Request (JSON)	`{ "owner": "0x742d35CCE634C0532925a3b8D4C9db964b4d8b6", "inputs": [ { "batchId": "123", "quantity": 500 }, { "batchId": "124", "quantity": 300 } ], "newBatchData": { ... } }`
Response (200)	`{ "success": true, "data": { "consumedInputs": [ ... ], "newBatch": { ... } }, "meta": { ... } }`
Flusso operativo	1. Validazione input (owner, inputs array, newBatchData) 2. Verifica esistenza batch input e stati (solo GREEN non consumato/distrutto) 3. Controllo ownership e balance on-chain per ogni input batch 4. Upload metadati IPFS per il nuovo lotto con allegati 5. Esecuzione transform on-chain (burn input + mint nuovo batch) 6. Creazione nuovo batch con relazioni padre-figlio e OperationType.TRANSFORM 7. Aggiornamento input batches come parzialmente consumati 8. Persistenza nuovo batch e aggiornamenti in MongoDB 9. Response al client con dettagli operazione e metadati gas
Errori tipici	400 input non valido, saldo insufficiente, batch non trovati 403 permessi insufficienti o batch non in stato valido per transform 404 uno o più batch input non trovati 500 errore blockchain, IPFS o persistenza 400 batch contaminati (RED) o completamente consumati/distrutti

Figura 31: Transform di un batch

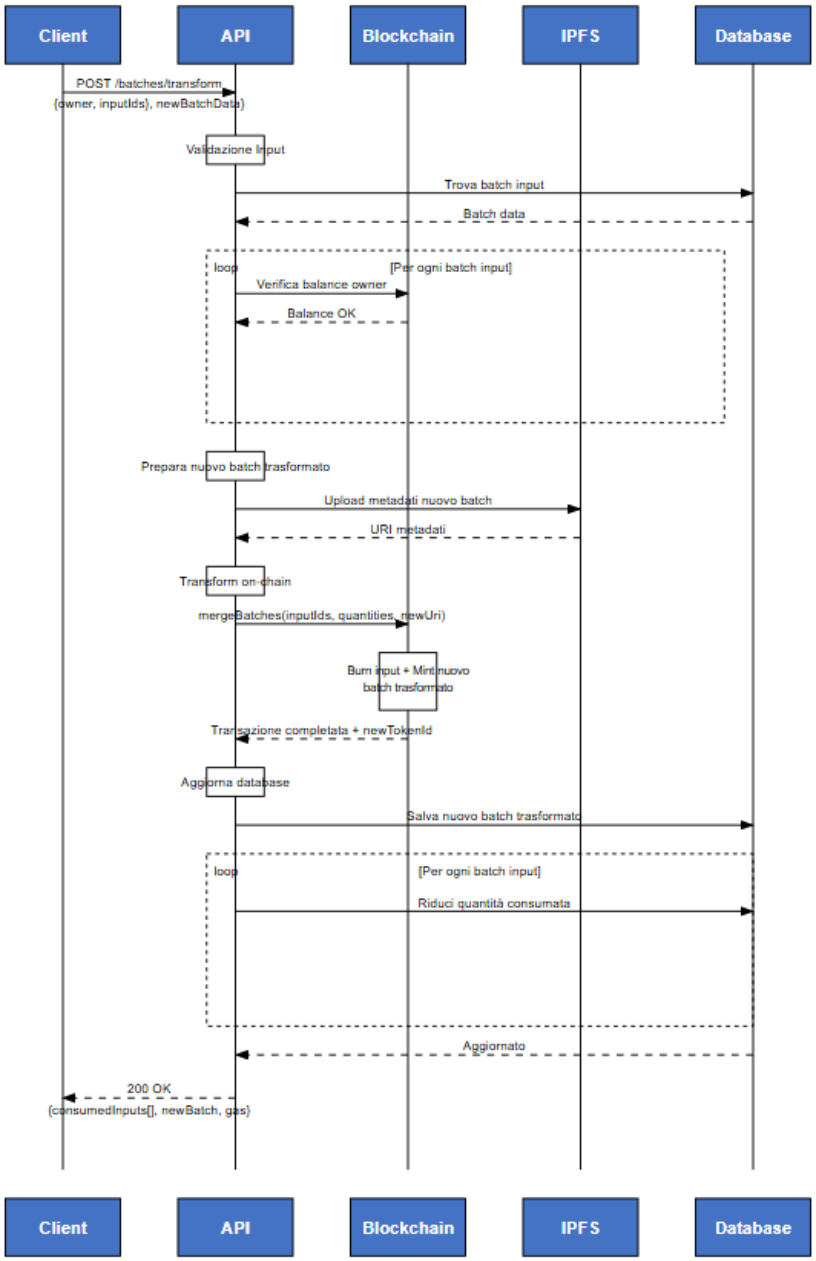


Figura 32: Diagramma di sequenza per il Transform

La Fig. 32 illustra, in forma di sequenza, il flusso dell'endpoint di transform: il client invia una richiesta HTTP POST /supply-chain/batches/transform con corpo JSON contenente l'indirizzo del proprietario, l'elenco dei batch in ingresso con le quantità da trasformare e la definizione del nuovo lotto (newBatchData). All'arrivo della richiesta, l'API Layer effettua il parsing del payload e avvia le procedure di validazione, verificando l'esistenza dei batch originali, il loro stato applicativo e la disponibilità delle quantità dichiarate. Superata questa fase, i servizi applicativi recuperano i dati dei batch, controllano i saldi on-chain dei token corrispondenti e si assicurano che il proprietario disponga delle quantità necessarie per eseguire la trasformazione. Una volta verificati i requisiti, i metadati del nuovo lotto vengono preparati e caricati su IPFS, ottenendo un URI immutabile.

In seguito, viene invocata l'operazione di transform sullo smart contract ERC-1155, implementata come chiamata alla funzione mergeBatches, poiché il transform è stato

modellato come operazione n a 1. In questo modo la blockchain esegue in maniera atomica il burn delle quantità consumate dei batch di ingresso e il mint del nuovo token trasformato con URI IPFS dedicato. La blockchain restituisce quindi l'esito della transazione (hash, numero di blocco ed eventi generati), che viene propagato ai servizi applicativi. Il livello servizi aggiorna di conseguenza lo stato del database: i batch genitori vengono ridotti della quantità consumata o marcati come esauriti, mentre il nuovo batch viene creato con i relativi attributi iniziali (quantità, unità di misura, stato GREEN/UNUSED, relazioni padre-figlio). Completato l'aggiornamento, l'API Layer costruisce la risposta finale al client con stato 200 OK, includendo l'elenco dei batch consumati, i dati del nuovo batch e i riferimenti all'esecuzione on-chain con le informazioni di costo gas. L'implementazione attuale utilizza il merge on-chain per modellare il transform come caso specifico di n a 1; l'introduzione di un'operazione più generale di tipo n a m, che consenta di generare più lotti trasformati a partire da più input, rappresenta un possibile sviluppo futuro del sistema.

Endpoint addizionali

È opportuno precisare che il prototipo implementa numerosi ulteriori endpoint, che per ragioni di chiarezza espositiva non vengono qui dettagliati. Si pensi, ad esempio, alle varianti di **transfer** che operano su insiemi di lotti (transfer collettivo), o agli endpoint dedicati all'esplorazione delle relazioni genealogiche (recupero dei *parent* o dei *child batches* di un determinato lotto). Tali funzionalità arricchiscono la piattaforma ma non sono centrali per illustrare la logica di base del modello, che è stata presentata attraverso i principali endpoint unitari.

A complemento di questi, menzioniamo però tre endpoint ulteriori che rappresentano vere e proprie **estensioni concettuali del modello di tracciabilità**:

- **Burn**: consente al proprietario di eliminare definitivamente i propri lotti (bruciando i token ERC-1155 corrispondenti). Dal punto di vista operativo, ciò equivale a eliminare un batch divenuto corrotto o non più conforme, garantendo che l'operazione sia attestata e immutabilmente registrata.
- **Validate**: introduce un meccanismo di verifica incrociata che confronta i dati conservati nei diversi layer del sistema (IPFS, MongoDB e blockchain). In tal modo si accerta che le informazioni siano coerenti e non si siano verificate anomalie di sincronizzazione, rafforzando l'affidabilità complessiva del modello.
- **Contaminate**: rappresenta un'estensione sperimentale che consente di marcare i token come contaminati e di propagare tale stato lungo il grafo parent-child. Questo "cambio di colore" dei lotti non ha solo valenza dimostrativa, ma apre scenari complessi legati alla gestione di crisi e richiami di prodotto, come nei casi di contaminazioni alimentari o di sicurezza sanitaria.

Burn:

La gestione del ciclo di vita di un lotto deve prevedere la possibilità di ridurne la corrispondente disponibilità. Questa operazione è gestita tramite la funzione di **burn**, che consente di distruggere in modo permanente una data quantità del token corrispondente al lotto.

Può essere applicata in scenari come lo smaltimento di prodotti non più utilizzabili, la perdita di materia prima o la distruzione controllata di lotti compromessi. Dal punto di vista tecnico, il burn si realizza attraverso una chiamata al contratto ERC-1155 che esegue la funzione di distruzione dei token. L'API verifica preliminarmente l'esistenza del lotto, la validità della quantità richiesta e il saldo on-chain dell'owner, quindi procede con l'esecuzione atomica dell'operazione. Il risultato è una riduzione della quantità residua del batch e l'aggiornamento del suo stato di ciclo di vita, che passa a PARTIALLY_DESTROYED o DESTROYED a seconda dei casi. La tabella in *Fig. 33* mostra la specifica dell'endpoint dedicato all'operazione di burn.

Sezione	Contenuto semplificato
Endpoint	Method: DELETE Path: /supply-chain/batches/{batchId} Content-Type: application/json
Descrizione	Permette di bruciare o distruggere una quantità specifica di token associati a un lotto esistente. L'operazione riduce la quantità del batch e aggiorna il suo ciclo di vita, garantendo la tracciabilità on-chain della distruzione.
Request (JSON payload)	`{ "quantity": 500, "owner": "0x742d35CCE634C0532925a3b8D4C9db964b4d8b6" }`
Response (200 OK)	`{ "success": true, "data": { "batch": { "batchId": "123", "currentQuantity": 200, "contaminationStatus": "GREEN", "lifecycleStatus": "PARTIALLY_DESTROYED" }, "burnedQuantity": 500, "remainingQuantity": 200, "transactionHash": "0xdef45e...", "blockNumber": 12346 }, "meta": { "txhash": "0xdef45e...", "blockNumber": 12346, "totalTimeMs": 1500, "chainMs": 1200, "mongoMs": 300 } }`
Flusso operativo	1. Validazione della richiesta (quantity > 0, batch esistente) 2. Verifica permessi burn (consentito su qualsiasi stato se esiste quantità disponibile) 3. Risoluzione dell'owner 4. Controllo del balance on-chain dell'owner 5. Esecuzione della funzione destroyBatch sul contratto ERC-1155 6. Aggiornamento quantità del lotto 7. Aggiornamento stato del ciclo di vita (PARTIALLY_DESTROYED o DESTROYED) 8. Registrazione dell'operazione nei metadati 9. Persistenza aggiornamenti su MongoDB 10. Restituzione della risposta al client
Errori tipici	• 400: quantità non valida o saldo insufficiente • 404: batch non trovato • 400: owner non risolto o chiave non valida • 500: errore blockchain o persistenza • 400: batch già completamente distrutto

Figura 33: Burn di un batch

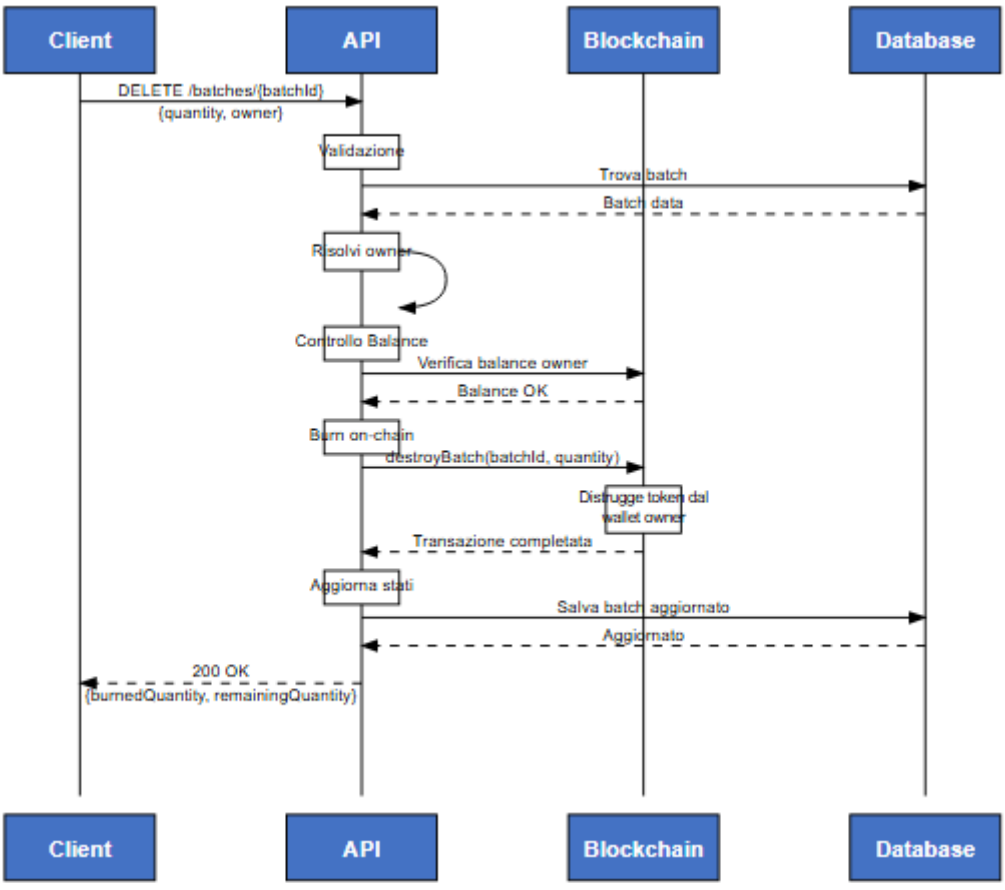


Figura 34: Diagramma di sequenza per il Burn

La Fig. 34 illustra, in forma di sequenza, il flusso dell'endpoint di burn: il client invia una richiesta HTTP DELETE /supply-chain/batches/{batchId} con corpo JSON contenente la quantità da bruciare e l'indirizzo del proprietario. All'arrivo della richiesta, l'API Layer effettua il parsing del payload e avvia le procedure di validazione, verificando l'esistenza del batch, la correttezza della quantità richiesta e la possibilità di applicare l'operazione. Una volta superati i controlli preliminari, il sistema risolve l'owner effettivo e interroga la blockchain per verificare il saldo disponibile del token associato al batch. Se l'owner dispone di una quantità sufficiente, viene invocata la funzione destroyBatch dello smart contract ERC-1155, che riduce in maniera atomica il saldo dei token sul wallet e registra la transazione. La blockchain restituisce quindi l'esito (hash, numero di blocco ed eventuali eventi generati), che viene propagato ai servizi applicativi. A questo punto il livello servizi aggiorna lo stato del database: la quantità del batch viene ridotta e il lifecycleStatus modificato in PARTIALLY_DESTROYED o DESTROYED a seconda dei casi.

Completato l'aggiornamento, l'API Layer costruisce la risposta finale al client con stato 200 OK, includendo la quantità bruciata, quella residua e i riferimenti all'esecuzione on-chain. L'operazione di burn consente dunque di gestire la fase finale del ciclo di vita di un lotto, garantendo che la distruzione dei token sia tracciata in modo coerente e immutabile sia on-chain sia nel database applicativo.

Validate:

Tra le estensioni concettuali del modello di tracciabilità, l'operazione di validate *(Fig. 35)* riveste un ruolo di rilievo perché permette di verificare l'affidabilità complessiva del sistema, verificando che i dati memorizzati nei diversi layer (blockchain, database e IPFS) siano coerenti e non abbiano subito alterazioni o perdite di sincronizzazione. In pratica, questo endpoint esegue una validazione incrociata su tutti i lotti, controllando l'integrità dei token, la corrispondenza degli URI e la consistenza dei metadati decentralizzati.

Dal punto di vista tecnico, il validate combina tre controlli principali:

1. **Token Integrity**, per accertarsi che il token esista on-chain.
2. **URI Match**, che confronta l'URI registrato on-chain per il token-id con quello presente nel database per il corrispettivo lotto.
3. **IPFS Data Match**, che scarica i metadati da IPFS e li confronta con quelli salvati in MongoDB, individuando eventuali discrepanze dovute a corruzioni o manomissioni.

L'esito della validazione viene restituito al client sotto forma di report che distingue batch validi, invalidi o in errore, consentendo agli attori della supply chain di avere uno strumento per monitorare la consistenza dei dati. L'endpoint validate rappresenta dunque uno dei possibili modi per implementare il controllo di integrità; il modello può essere esteso con ulteriori modalità di validazione, ad esempio focalizzate su singoli tokenId o su scenari di audit più approfonditi, in funzione del livello di sicurezza e trasparenza che si intende raggiungere.

Sezione	Contenuto semplificato
Endpoint	Method: GET Path: /supply-chain/batches/validate Content-Type: application/json
Descrizione	Permette di validare tutti i lotti esistenti nel sistema contro i dati on-chain e IPFS. L'endpoint verifica l'integrità dei token, la corrispondenza degli URI e la coerenza dei dati pubblici tra database, blockchain e IPFS, garantendo la tracciabilità completa del sistema.
Request (Query Parameters)	`GET /supply-chain/batches/validate?detailed=true` detailed (optional): boolean Se true, restituisce dettagli completi per ogni lotto
Response (200 OK)	`{ "success": true, "data": { "summary": { "total": 3, "valid": 3, "invalid": 0, "errors": 0 }, "results": [{ "batchId": "123", "status": "valid", "checks": { "tokenIntegrity": "VALID_EXISTS", "uriMatch": true, "ipfsMatch": { "search": true, "details": [ ] } }, { "batchId": "124", "status": "valid", "checks": { "tokenIntegrity": "VALID_MINTED", "uriMatch": true, "details": [ ] } }, { "batchId": "125", "status": "valid", "checks": { "tokenIntegrity": "VALID_EXISTS", "uriMatch": true, "ipfsMatch": true, "details": [ ] } }], "meta": { "totalTime": 2500, "chainMs": 1800, "ipfsMs": 700 } } }`
Flusso operativo	1. Recupero di tutti i lotti dal database MongoDB 2. Verifica del balance on-chain del token 3. Recupero dell'URI del token dalla blockchain 4. Validazione corrispondenza URI tra blockchain e database 5. Se URI corrisponde, recupero e confronto dati IPFS 6. Determinazione stato finale di validazione 7. Calcolo del riepilogo (totale, validi, non validi, errori) 8. Restituzione dei risultati con metadati di timing
Errori tipici	• 500: Errore interno del server durante la validazione • 500: Errore di connessione alla blockchain • 500: Errore di accesso ai dati IPFS • 500: Errore di accesso al database MongoDB **Stati di validazione:** • **Valid:** Token esiste, URI corrisponde, dati IPFS coerenti • **Invalid:** Token esiste ma URI o dati IPFS non corrispondono • **Error:** Errore durante la validazione (network, blockchain, IPFS)

Figura 35: Validate di batches

La *Fig. 36* illustra il flusso dell'endpoint di validate: il client invia una richiesta HTTP GET /supply-chain/batches/validate e l'API Layer recupera dal database la lista dei lotti registrati. Per ciascun batch, il sistema avvia tre controlli principali.

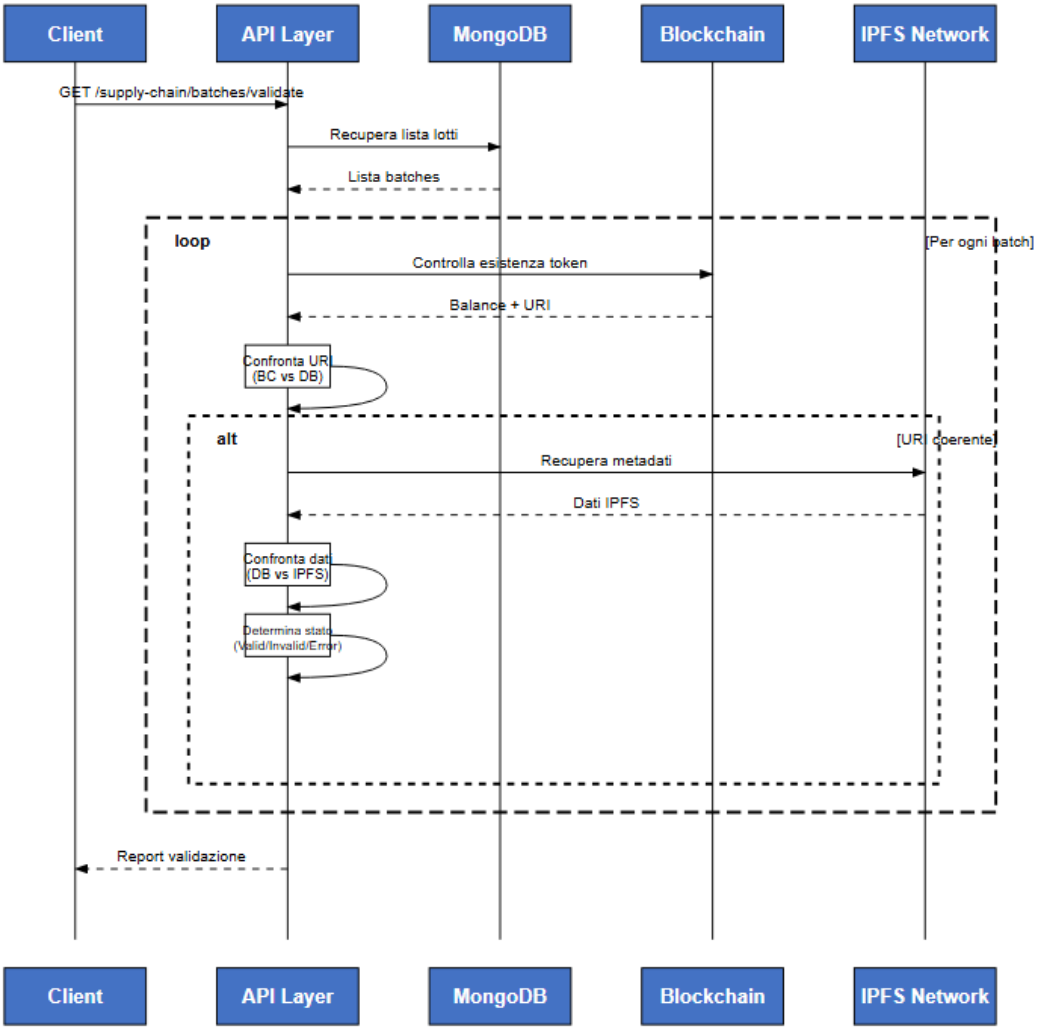


Figura 36: Diagramma di sequenza di Validate

Il primo è un check di esistenza del token on-chain, che confronta la quantità rilevata dalla blockchain con quella riportata nel documento aggregato conservato in MongoDB. Il secondo controllo verifica che l'URI registrato sulla blockchain coincida con quello persistito nel database, assicurando l'allineamento tra i due livelli. Infine, se l'URI è coerente, viene interrogato IPFS per recuperare i metadati decentralizzati e confrontarli con i dati pubblici conservati off-chain, in modo da rilevare eventuali manomissioni o corruzioni. Al termine di questi passaggi, per ogni batch viene determinato uno stato finale (valid, invalid o error) e l'API genera un report riepilogativo che distingue i lotti correttamente validati da quelli che presentano incongruenze. È opportuno sottolineare che questa implementazione del validate è stata sviluppata per dimostrare la fattibilità della validazione incrociata e testata in scenari in cui i lotti venivano trasferiti interamente, senza suddivisioni di un singolo specifico lotto/batch/token tra più proprietari. La logica attuale si basa infatti sul bilancio aggregato di un utente; per supportare casi più complessi, in cui un token risulti frammentato tra più wallet, sarebbe necessario introdurre un algoritmo di ricostruzione delle quantità distribuite o mantenere un tracciamento puntuale dei bilanci per ciascun owner. L'implementazione di tali estensioni rappresenta un possibile sviluppo futuro per rendere la validazione più completa e robusta.

Contaminate:

Tra gli endpoint aggiuntivi del prototipo è stato introdotto l'endpoint contaminate, una funzionalità extra ispirata al paper Cerere, nel quale viene mostrato un tipico incidente di tracciabilità alimentare e la conseguente propagazione della contaminazione lungo la filiera (Fig. 37). Questo scenario evidenzia come un singolo lotto compromesso possa influenzare tutti i lotti collegati, sia a monte (upstream) che a valle (downstream), e come sia fondamentale disporre di strumenti che permettano una propagazione controllata e tracciata dello stato di contaminazione. Nel prototipo assumiamo esplicitamente l'individuazione di un lotto contaminato in una posizione intermedia della supply chain; per questo, rispetto al modello di Cerere, distinguiamo due livelli operativi: RED per i lotti sicuramente contaminati (il lotto sorgente e tutti i suoi discendenti) e YELLOW per i lotti potenzialmente contaminati (gli antenati del lotto sorgente e i loro discendenti, ad esclusione del lotto sorgente e del suo downstream). In presenza di conflitto vale la precedenza RED su YELLOW. Per rispondere a questa esigenza, l'endpoint contaminate consente di marcare come contaminato un lotto specifico e propagare automaticamente la contaminazione a tutti i lotti correlati. L'operazione viene eseguita con un approccio batch, aggiornando simultaneamente gli stati di tutti i token interessati tramite un'unica transazione on-chain. La *Fig. 38* riassume le principali caratteristiche di questo endpoint.

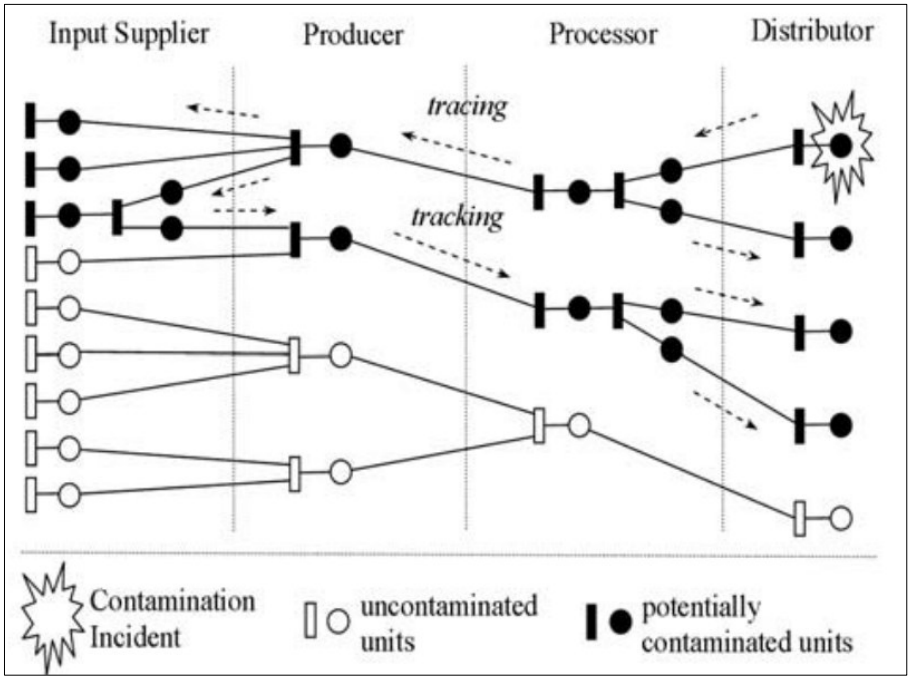


Figura 37: Incidente di contaminazione

Sezione	Contenuto semplificato
Endpoint	**Method:** POST **Path:** /supply-chain/batches/{batchId}/contaminate **Content-Type:** application/json
Descrizione	Permette di contaminare un lotto specifico e propagare automaticamente la contaminazione a tutti i lotti correlati (upstream e downstream) nella catena di fornitura. Il sistema applica stati di contaminazione (RED per contaminati, YELLOW per potenzialmente contaminati) e sincronizza i cambiamenti on-chain con una singola transazione batch.
Request (JSON payload)	`{ "reason": "contamination detected" }`
Response (200 OK)	`{ "success": true, "data": { "message": "Contamination cascade completed", "txHash": "0xabc123...", "totalProcessed": 6, "contaminatedRED": ["123", "124", "125"], "upstreamYELLOW": ["120", "121", "122"] }, "meta": { ... } }`
Flusso operativo	1. **Validazione richiesta** (batchId esistente, permessi contaminazione) 2. **Calcolo propagazione** (analisi grafo upstream/downstream) 3. **Filtro lotti da aggiornare** (solo quelli che cambiano stato) 4. **Transazione batch** (setContaminationStatus con tutti i token) 5. **Aggiornamento database** (stati contaminazione in MongoDB) 6. **Log operazioni atomiche** (tracciamento cambi stato) 7. **Propagazione automatica** (RED downstream, YELLOW upstream) 8. **Restituzione risultati** (lotti processati e stati applicati)
Errori tipici	• **404:** Batch non trovato • **400:** Batch non può essere contaminato (stato non GREEN) • **500:** Errore propagazione contaminazione • **500:** Errore transazione blockchain • **500:** Errore aggiornamento database
Stati di contaminazione	• **GREEN:** Lotto sicuro (stato iniziale) • **YELLOW:** Potenzialmente contaminato (upstream/siblings) • **RED:** Contaminato (target + downstream) • **Propagazione:** RED si propaga downstream, YELLOW upstream

Figura 38: Contaminate di un batch

Il cuore della funzionalità è l'algoritmo di propagazione della contaminazione, che opera sul grafo delle relazioni tra i lotti collegati al lotto contaminato: ogni nodo rappresenta, cioè, un lotto, mentre gli archi definiscono le relazioni padre-figlio (*Fig. 39*) dovute alle operazioni di filiera. L'algoritmo utilizza due ricerche BFS in avanti (forward) e una all'indietro (backward): la prima forward identifica i discendenti del lotto iniziale *(getChildren, Fig. 40)*, che devono essere marcati come contaminati (RED); la backward individua gli antenati del lotto iniziale *(getParents, Fig. 40)*; la seconda forward *(getChildrenParents, Fig. 40)*, applicata a tali antenati, trova i loro discendenti (ad eccezione del lotto sorgente e del suo downstream), che vengono marcati come potenzialmente contaminati (YELLOW). Il lotto iniziale e i suoi discendenti diventano RED (contaminati), mentre gli antenati e i loro discendenti, ad esclusione del downstream del lotto sorgente della contaminazione, vengono classificati come YELLOW (potenzialmente contaminati), a condizione che si trovino ancora in stato GREEN. In caso di conflitto, vale la regola di precedenza RED su YELLOW, così da garantire che un lotto compromesso non venga mai declassato a potenzialmente contaminato. In questo modo si mantiene la coerenza dello stato della catena, evitando di riclassificare lotti già compromessi.

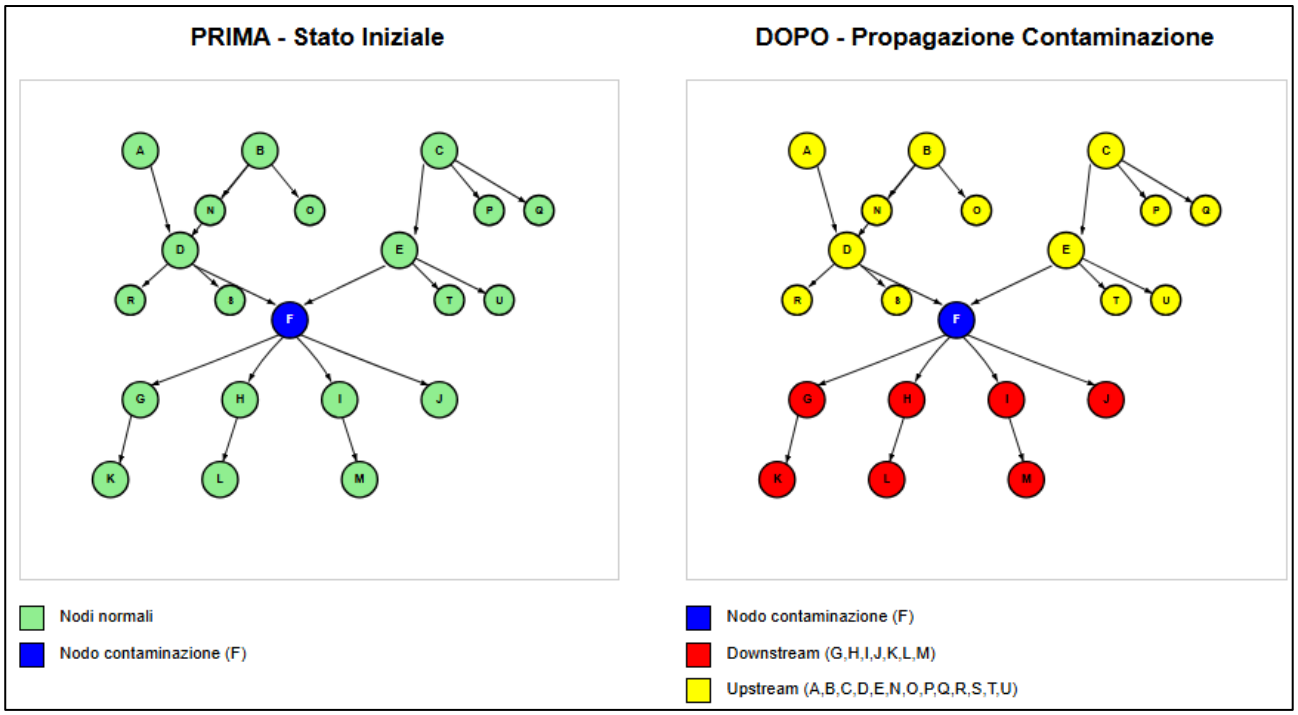


Figura 39: Esempio di Contaminazione

Per evitare transazioni ridondanti, l'algoritmo filtra esclusivamente i lotti che cambiano effettivamente stato *(getStateBulk())* e applica l'aggiornamento con una chiamata on-chain unica. Subito dopo l'esecuzione della transazione, lo stato dei lotti viene aggiornato nel database e viene registrata un'operazione che conserva traccia della modifica ai fini della tracciabilità. Il diagramma di sequenza in *Fig. 40* illustra il flusso operativo dell'endpoint.

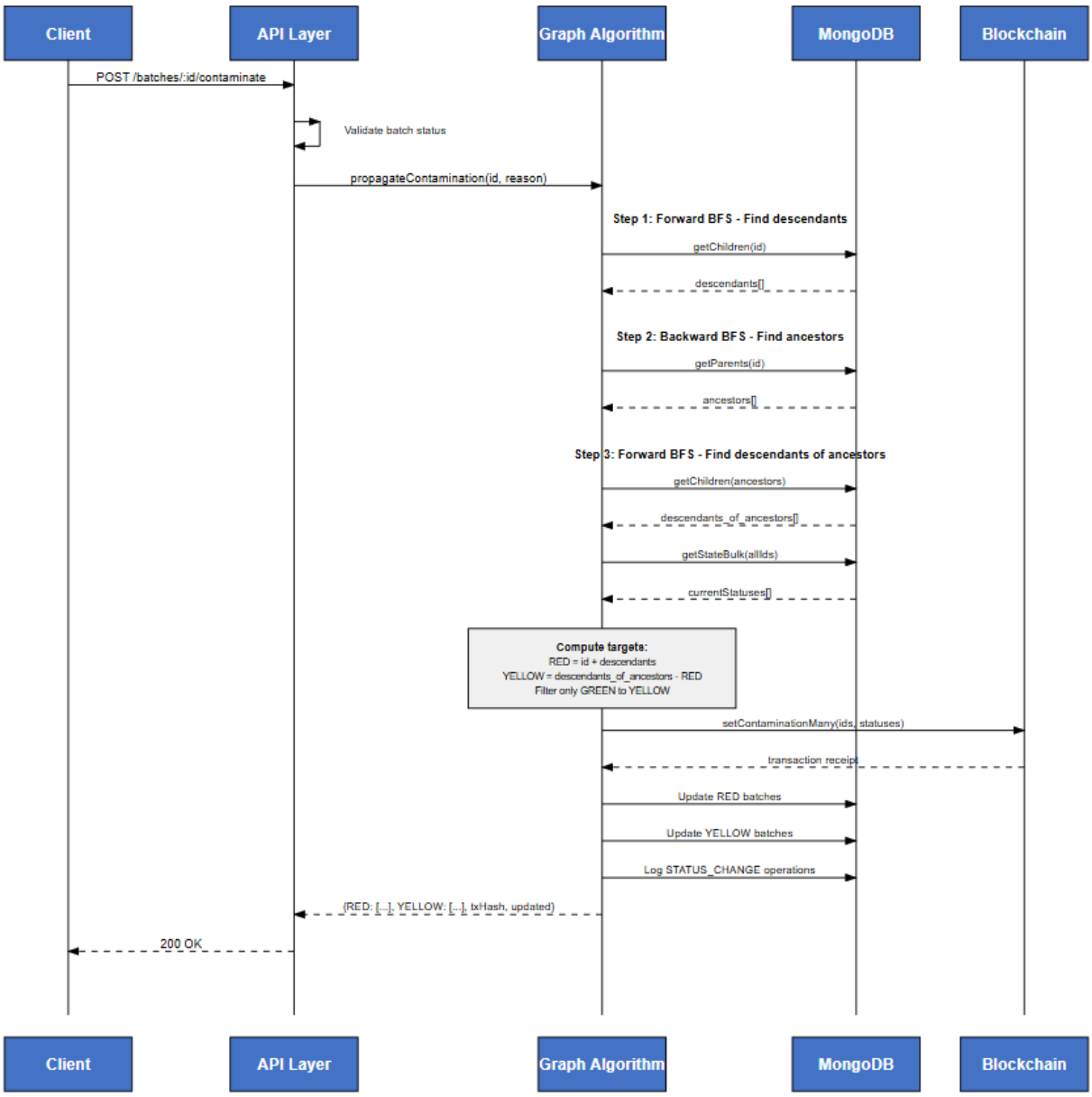


Figura 40: Diagramma di sequenza di Contaminate

Il capitolo ha descritto i principali flussi operativi implementati tramite API, mostrando come viene gestito il ciclo di vita dei lotti e le estensioni introdotte a supporto di possibili scenari avanzati. Sono stati analizzati gli endpoint di creazione, trasferimento, suddivisione, fusione e trasformazione, insieme a operazioni aggiuntive come burn, validate e contaminate. Ogni endpoint è stato presentato come parte di un modello coerente, basato su principi REST e sull'orchestrazione coordinata tra blockchain, IPFS e database, in modo da garantire immutabilità, persistenza e tracciabilità end-to-end. La distinzione tra operazioni multi-lotto (collection level) e puntuali (resource level) consente inoltre di gestire in maniera flessibile sia le trasformazioni complesse sia le interazioni elementari. Questo quadro completa il passaggio dalla dimensione architetturale a quella applicativa, mostrando come i processi siano effettivamente implementati e resi disponibili attraverso un'interfaccia uniforme. Nel capitolo successivo verranno presentati i casi di test e l'analisi dei costi e delle prestazioni, con l'obiettivo di valutare l'efficienza e la scalabilità del modello.

Capitolo 7: Validazione e benchmark

La progettazione della suite di test ha seguito un approccio pensato per garantire una copertura sia dal punto di vista funzionale che operativo. I test sono stati concepiti come strumenti in grado di verificare l'efficacia degli endpoint e al contempo di fornire visibilità sulle cause di eventuali anomalie, intese come tutte le deviazioni dal comportamento atteso del sistema indipendentemente dalla componente che le origina. Queste possono derivare da errori applicativi, come input non validi o stati incoerenti; da violazioni delle regole di business, ad esempio tentativi di trasferire o fondere lotti con stato non consentito; oppure da malfunzionamenti di servizi esterni, quali IPFS, blockchain o database.

Ogni categoria di test è stata progettata per rilevare e distinguere queste condizioni, generando report e log dettagliati che consentono di individuare rapidamente la componente responsabile e la natura dell'anomalia. Per questo motivo ogni categoria di test include logging dettagliato: i report distinguono chiaramente se un fallimento dipende da errori applicativi, da conflitti nelle regole di business, o da malfunzionamenti di servizi esterni (IPFS, blockchain, database). Tale strategia consente di validare gli endpoint e di ricondurre rapidamente l'origine di un problema alla componente responsabile.

Definizione e analisi dei test case

L'intero sistema di test è articolato in tre blocchi principali: unit test, end-to-end test e test di contaminazione.

Unit Test

Gli unit test hanno lo scopo di validare **ogni singolo endpoint** esposto dall'API, assicurando che il comportamento sia conforme alle specifiche. Ogni operazione è stata testata separatamente con dati sia validi che non validi, in modo da coprire i casi di successo e di errore. Gli endpoint sottoposti a test sono:

- **POST /batches**: creazione di un nuovo batch, controllo della corrispondenza tra i dati salvati in IPFS, MongoDB e blockchain e verifica dell'inizializzazione corretta degli stati qualitativo e quantitativo.

- **POST /batches/:id/transfer**: trasferimento di proprietà, con controllo sull'esecuzione dell'operazione consentito solo su batch *GREEN* e aggiornamento di ownership.

- **POST /batches/:id/split**: suddivisione di un lotto in più discendenti, con verifica della conservazione delle quantità totali e della corretta generazione dei riferimenti in IPFS e database.

- **POST /batches/merge**: fusione di più batch, controllo della somma delle quantità e della propagazione corretta degli stati.

- **POST /batches/transform**: trasformazione di un batch in un prodotto derivato, con validazione della relazione padre-figlio e dei metadati associati.

- **POST /batches/:id/contaminate**: marcatura di un batch come contaminato, verifica della propagazione dello stato secondo le regole del sistema.

- **DELETE /batches/:id**: distruzione di un batch, testata in ogni stato (incluso contaminato), per confermare che l'operazione sia sempre permessa e che le dipendenze siano aggiornate.

Oltre alle regole specifiche, ogni unit test controlla la **coerenza dei dati** tra i sistemi coinvolti (blockchain, IPFS, MongoDB), validando che gli hash e i riferimenti siano sincronizzati.

Test end-to-end

I test end-to-end (E2E) simulano scenari realistici di supply chain, concatenando più operazioni in sequenza. A differenza degli unit test, l'obiettivo non è isolare il singolo endpoint, ma verificare che il sistema mantenga **consistenza globale** lungo l'intero flusso operativo. La sequenza di operazioni segue un ordine simulato di ciclo di produzione:

- **CREATE**: creazione di lotti di materie prime (batch iniziali).

- **TRANSFER**: trasferimento a un nuovo attore della filiera, verificando che solo batch validi possano essere spostati.

- **SPLIT**: divisione in sotto-lotti, per simulare prodotti destinati a diverse destinazioni.

- **MERGE**: fusione di lotti provenienti da fonti differenti.

- **TRANSFORM**: trasformazione dei lotti in un prodotto derivato intermedio o finale.

- **CONTAMINATE**: marcatura di un lotto intermedio per testare la propagazione a valle e a monte.

- **BURN (DELETE)**: eliminazione dei residui per concludere il ciclo.

L'ordine è stato scelto per rappresentare uno scenario **industriale realistico**: dalla materia prima al prodotto finale, passando attraverso trasformazioni e passaggi di proprietà. In questo contesto si verifica che gli stati dei batch siano sempre aggiornati coerentemente, che le quantità siano preservate e che la contaminazione si propaghi lungo l'intero grafo di dipendenze.

Test di contaminazione

Una categoria separata di test è stata dedicata agli scenari di contaminazione. In questo caso i batch sono organizzati in **grafi** con convergenze e diramazioni. Lo scopo è validare la logica di propagazione, verificando che:

- lo stato *YELLOW* venga propagato a monte (upstream),

- lo stato *RED* si propaghi a valle (downstream),

- in caso di conflitti lo stato *RED* prevalga su *YELLOW*,

- i batch discendenti e ascendenti mantengano coerenza dopo ogni aggiornamento.

Questi test dimostrano che le regole di contaminazione sono efficaci anche in presenza di strutture come quelle tipiche di una filiera con più trasformazioni intermedie e convergenze di diversi lotti.

Obiettivi dei test

L'attività di testing consente di verificare, in maniera sistematica, quattro dimensioni fondamentali del sistema:

- la **correttezza locale** di ciascun endpoint, valutata rispetto alle regole di business e agli stati ammessi;

- la **consistenza globale** dei workflow end-to-end, che garantisce il mantenimento della tracciabilità lungo l'intera filiera;

- la **sincronizzazione dei dati** tra blockchain, IPFS e database, indispensabile per assicurare integrità e non ripudiabilità.

- la **propagazione** delle regole di contaminazione nei grafi di dipendenze;

Complessivamente forniscono la dimostrazione che il modello implementato preserva integrità e tracciabilità in tutti gli scenari considerati. Nella sezione successiva l'attenzione si sposta dalla dimensione logica a quella quantitativa, attraverso i risultati di benchmarking che analizzano tempi di esecuzione e costi in gas, con l'obiettivo di individuare colli di bottiglia e valutare l'impatto pratico dell'architettura.

Benchmarking

Gli script sviluppati permettono di analizzare le prestazioni e i costi associati alle principali operazioni del sistema di tracciabilità. Essi sono stati implementati in **Node.js** e integrano, all'interno degli endpoint, un sistema di raccolta di metadati capace di restituire per ogni invocazione non solo l'esito funzionale, ma anche una scomposizione dei tempi di esecuzione e dei consumi. In particolare, ogni chiamata viene misurata rispetto ai contributi principali che caratterizzano l'architettura: il tempo necessario alle interazioni con **IPFS** per il caricamento e il recupero dei metadati, quello richiesto per l'esecuzione della transazione **on-chain** e, infine, la quota attribuibile al database **MongoDB** per la persistenza e il recupero delle informazioni. In parallelo vengono raccolti i consumi di **gas**, con registrazione del gasUsed, del costo in Wei e della conversione corrispondente in ETH, così da associare alla dimensione prestazionale anche una valutazione economica. Gli scenari di prova realizzati coprono sequenze di operazioni sugli endpoint, quali creazione di lotti, merge, split, trasferimenti, trasformazioni, contaminazioni e distruzioni: ricostruiscono catene di approvvigionamento in cui i materiali vengono prodotti, trasferiti, trasformati, suddivisi, distribuiti, includendo la propagazione di contaminazioni.

Sono stati predisposti due script principali: il primo (*endpoint-bench*) esegue test mirati sugli endpoint elementari, misurando tempi e costi di ciascuna singola operazione; il secondo (*supplychain-bench*) ricostruisce invece scenari di filiera simulati, concatenando più operazioni in sequenza per riprodurre casi d'uso realistici di produzione, trasformazione e

distribuzione. È fondamentale sottolineare che tutte le misurazioni sono state condotte in un contesto locale: la blockchain è stata istanziata con **Ganache**, in cui la generazione dei blocchi è pressoché istantanea e la validità delle transazioni è considerata dopo un solo blocco (tx.await(1)), anche il database **MongoDB** è eseguito in locale, minimizzando così la latenza introdotta dal livello di persistenza. Tali condizioni rendono i risultati ottimistici in termini di tempi assoluti, ma al contempo utili per l'analisi comparativa tra operazioni differenti. Gli script producono come output un file in formato **CSV**, nei quali vengono riportati per ogni step di test i tempi totali e scomposti, il gas consumato e il costo stimato in ETH. Poiché le prestazioni sono state misurate a livello di singolo endpoint in entrambi gli scenari, i risultati sono stati **aggregati in un unico dataset**, così da ottenere una vista sintetica e comparabile dell'intero sistema. La *Fig. 41* riporta tale aggregazione, presentando per ciascun endpoint il numero di campioni raccolti, i tempi medi (totali e suddivisi tra IPFS, blockchain e MongoDB), nonché i consumi medi di gas e il corrispondente costo stimato in ETH.

Endpoint	Count	Total Ms Mean	IPFS Ms Mean	Chain Ms Mean	Mongo Ms Mean	gasUsed_mean	gasCostETH_mean
DELETE /batches/:id	3	569.0	0.0	564.0	5.0	26691.0	0.000534
POST /batches	8	2138.9	816.6	1305.8	16.5	134401.0	0.002688
POST /batches/:id/contaminate	2	1531.0	0.0	826.5	704.5	51373.0	0.001027
POST /batches/:id/split	3	3606.0	1143.3	2451.7	11.0	239286.0	0.004786
POST /batches/:id/transfer	8	579.1	0.0	569.8	9.4	59271.5	0.001185
POST /batches/merge	5	2153.4	530.0	1617.2	6.2	143210.0	0.002864
POST /batches/transform	2	2366.0	507.0	1852.5	6.5	144191.0	0.002884

Figura 41: Endpoint con valori aggregati

Questa sintesi aggregata costituisce la base per le analisi discusse nei paragrafi successivi, in cui i valori vengono ulteriormente rielaborati in forma grafica e commentati.

Considerazioni sulle Prestazioni

Per la valutazione delle prestazioni del sistema di tracciabilità, sono stati misurati i **tempi medi di esecuzione** delle principali operazioni, scomponendo il contributo dei tre sottosistemi coinvolti. L'analisi è finalizzata a determinare non soltanto la durata complessiva delle operazioni, ma anche la **ripartizione della latenza** tra i diversi componenti dell'architettura, con l'obiettivo di identificare i fattori predominanti.

Il grafico di *Fig. 42* riporta il **tempo totale medio per endpoint**. Le misurazioni consentono di distinguere tre gruppi principali:

- **Operazioni a prestazioni ridotte (SPLIT, MERGE, TRANSFORM, CREATE)**, caratterizzate da tempi medi compresi tra 2 e 3,5 secondi, dovuti all'esecuzione concatenata di transazioni on-chain, caricamenti di metadati su IPFS e aggiornamenti consistenti in database.
- **Operazioni a prestazioni intermedie (CONTAMINATE)**, con tempi attorno a 1,5 secondi, riconducibili principalmente alla propagazione dello stato lungo il grafo delle dipendenze.
- **Operazioni a prestazioni elevate (TRANSFER, DELETE)**, con tempi inferiori a 600 ms, limitate a una singola transazione on-chain e a modifiche marginali lato database.

Il grafico di *Fig. 43* evidenzia la **distribuzione della latenza tra i sottosistemi**. Le misurazioni indicano che la **blockchain rappresenta il principale fattore di rallentamento**. I test sono stati condotti in ambiente locale con **Ganache**, in cui la generazione del blocco è istantanea e la validità della transazione è riconosciuta dopo un solo blocco (tx.await(1)). Queste condizioni determinano valori **ottimistici** rispetto a scenari reali: su una rete pubblica la latenza dipende dal **tempo medio di generazione del blocco**, dal **numero di conferme richieste** e dal grado di congestione, con tempi sensibilmente superiori. Il contributo di **IPFS** si manifesta in alcune operazioni, soprattutto in quelle che comportano la generazione di nuovi metadati, ma non costituisce l'elemento dominante nella latenza complessiva. **MongoDB**, al contrario, risulta sistematicamente trascurabile, confermando l'efficienza nei carichi considerati. L'analisi dei consumi di **gas**, che quantificano il costo computazionale e lo spazio occupato on-chain, conferma l'andamento osservato: le operazioni di **split e merge** sono sia le più lente sia le più onerose. Nel capitolo successivo tali costi e tempistiche saranno confrontati con i tempi di blocco tipici di **Ethereum** e con quelli di soluzioni **layer 2** come **Polygon**, al fine di delineare un quadro più realistico delle prestazioni attese in caso di distribuzione su reti pubbliche.

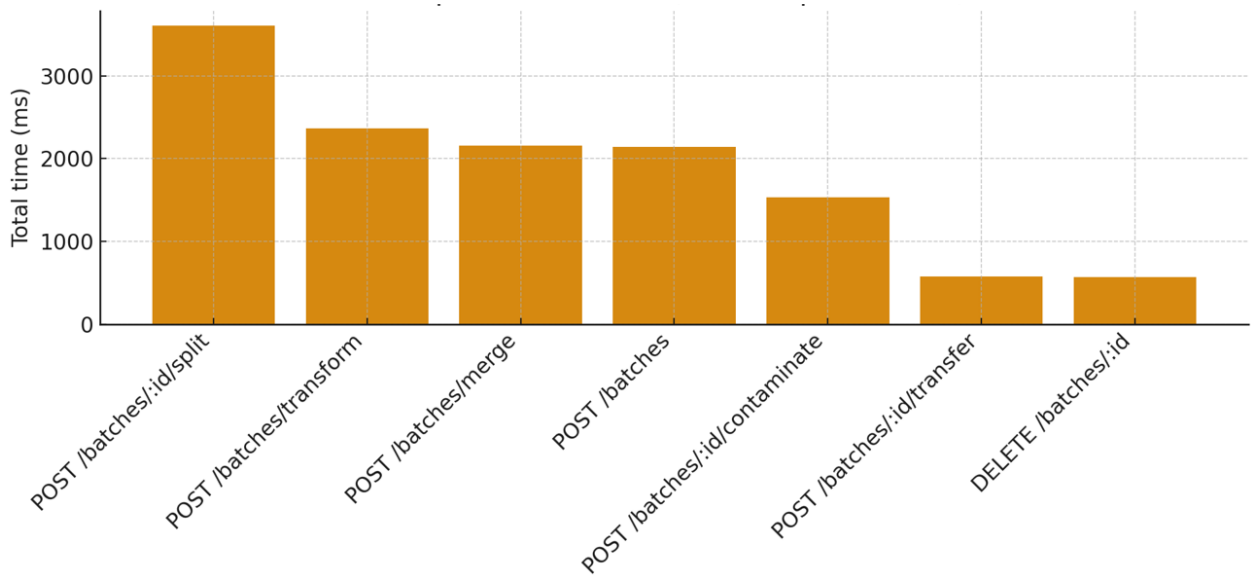


Figura 42: Tempo totale medio per Endpoint

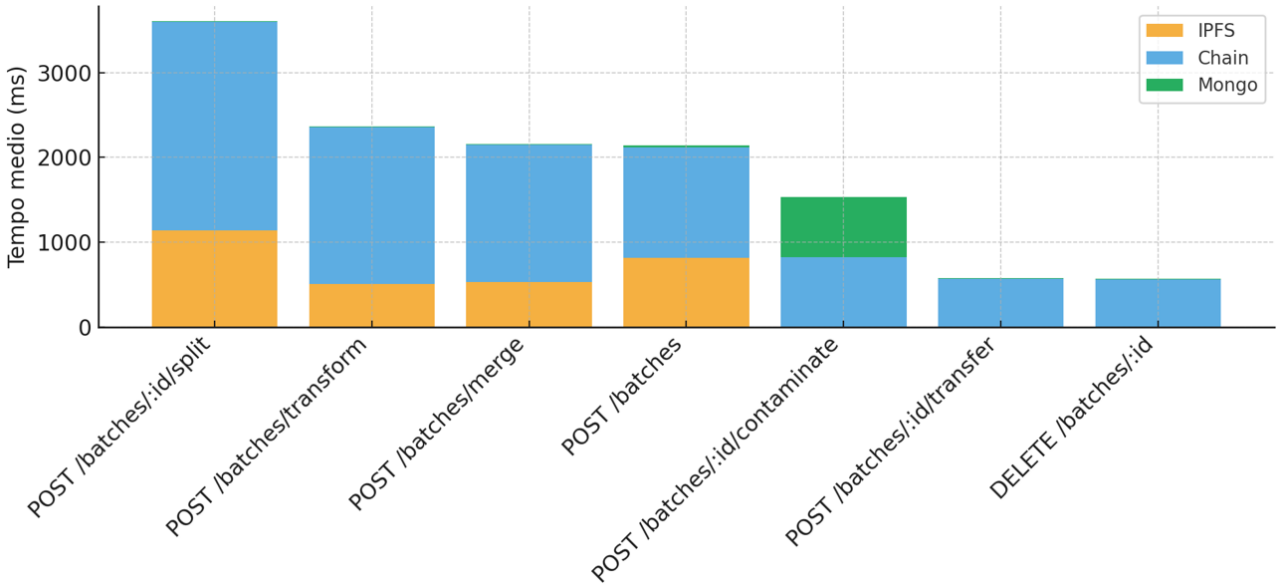


Figura 43: Breakdown dei tempi per Endpoint

Considerazioni sui Costi

Oltre all'analisi delle prestazioni, è stato eseguito uno studio dei consumi di gas associati. Il gas rappresenta il meccanismo con cui Ethereum quantifica il costo computazionale delle operazioni on-chain: ogni scrittura su storage, emissione di eventi o esecuzione di transazioni consuma una certa quantità di gas, che moltiplicata per il prezzo unitario determina il costo finale della transazione in ETH. La successiva conversione in valuta corrente consente di valutare in maniera diretta l'impatto economico delle scelte architetturali.

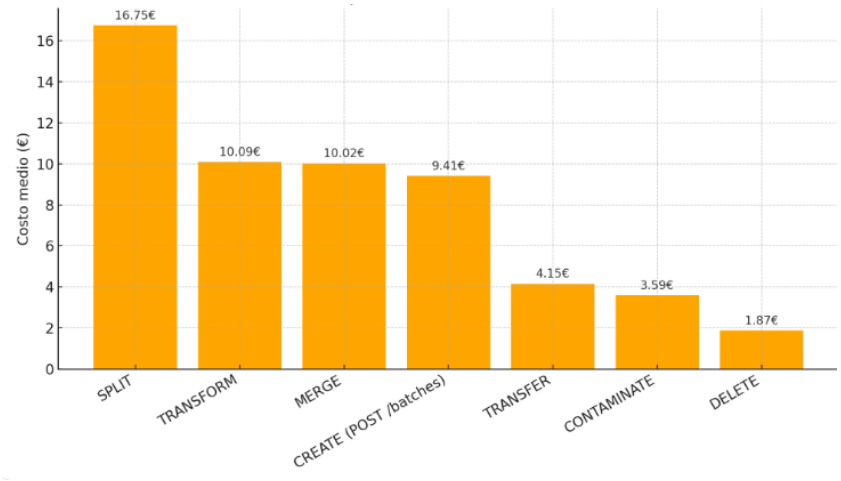


Figura 44: Comparazione costi operazioni

Il grafico di *Fig. 44* riporta i costi medi delle operazioni, calcolati al valore di ETH al 20 agosto 2025 (≈3553 €). È possibile distinguere tre fasce principali:

- **Operazioni a costo elevato (SPLIT, MERGE, TRANSFORM, CREATE)**: caratterizzate da spese comprese tra ~9,5 € e ~17 €, dovute alla combinazione di più azioni di persistenza su storage e aggiornamenti di stato on-chain. Queste funzioni creano nuovi identificativi di token ERC-1155, aggiornano i bilanci e soprattutto allocano nuove stringhe (URI dei batch) nello storage del contratto, operazione particolarmente onerosa in termini di gas.
- **Operazioni a costo intermedio (CONTAMINATE, TRANSFER)**: con valori tra ~3,5 € e ~4,2 €, in cui il peso principale è attribuibile all'aggiornamento dello stato on-chain senza l'introduzione di nuove stringhe. In questi casi si modificano mapping già esistenti, ad esempio aggiornando lo status di un token.
- **Operazioni a costo ridotto (DELETE/BURN)**: con un'incidenza economica più contenuta (~1,9 €). In questo scenario il saldo di un batch viene azzerato e, grazie al meccanismo di gas refund sugli slot riportati a zero, il costo effettivo della transazione si riduce. È il caso più economico perché non vengono allocate nuove stringhe né nuove strutture nello storage, ma solo aggiornati i bilanci.

Anche adottando strategie di ottimizzazione come l'utilizzo di transazioni batch, i costi rimangono **elevati e spesso proibitivi** su Ethereum mainnet. La persistenza e l'elaborazione on-chain comportano inevitabilmente un impegno economico significativo, che limita la scalabilità rispetto a scenari caratterizzati da frequente esecuzione di operazioni complesse. I costi osservati per le varie operazioni sono direttamente correlati al tipo di dati persistenti: operazioni come **CREATE** e **SPLIT** risultano onerose perché implicano nuove scritture di stringhe nello storage, mentre operazioni come **TRANSFER** o **CONTAMINATE** sono più leggere poiché aggiornano soltanto valori già presenti. Infine, operazioni come **DELETE/BURN** beneficiano del rimborso di gas per gli slot azzerati. Va sottolineato che i costi potrebbero essere ulteriormente ottimizzati: i valori presentati includono infatti anche la memorizzazione di uno stato che è un parametro per noi addizionale e che, se rimosso o sostituita da eventi/log, permetterebbe una diminuzione significativa dei consumi di gas. Nelle prossime pagine verrà presentato un confronto con i costi e le tempistiche ottenibili adottando soluzioni **Layer 2**, come **Polygon**, che consentono di ridurre drasticamente l'incidenza economica mantenendo inalterate le proprietà di sicurezza e tracciabilità.

Capitolo 8: Estensioni, Criticità e sviluppi futuri

Dopo aver descritto l'architettura del sistema, i flussi operativi e i risultati dei benchmark, è opportuno soffermarsi sulle possibili estensioni e criticità che emergono dall'analisi di quanto sviluppato. L'approccio basato su token ERC-1155 e su un'infrastruttura distribuita, ha dimostrato la fattibilità di un modello di tracciabilità tokenizzato per la supply chain, ma l'adozione su larga scala evidenzia sfide che richiedono ulteriori ottimizzazioni. Questo capitolo affronta quindi alcune delle principali direzioni di sviluppo e le problematiche ancora aperte, con l'obiettivo di delineare una visione critica del sistema. In particolare, l'attenzione si concentra su:

- l'integrazione di **soluzioni di Layer 2** (come Polygon), per ridurre drasticamente i tempi e i costi delle transazioni, mantenendo le proprietà di sicurezza della blockchain;
- la **gestione delle chiavi utente**, con una menzione ai principali approcci che si potrebbero adottare: custodial e non-custodial;
- la generazione degli identificativi dei lotti;
- la possibile introduzione di **meccanismi di firma congiunta**, intesi come firma sia del mittente sia del ricevente su una stessa transazione, per certificare in modo univoco la validità di trasferimenti e acquisizioni ed evitare contestazioni.

In questo modo, il capitolo mira a evidenziare come le scelte tecnologiche e architetturali possano evolvere per affrontare i limiti di costo, scalabilità e sicurezza, aprendo la strada a un'adozione più ampia del modello in scenari reali.

Soluzioni Layer 2

L'elevato costo delle operazioni di scrittura nello storage del contratto e la latenza intrinseca della rete, legata al tempo medio di generazione dei blocchi (≈15 secondi [20]), rendono particolarmente onerose operazioni complesse quali split, merge e transform. Le soluzioni di Layer 2 sono state sviluppate proprio per mitigare tali criticità, introducendo meccanismi che permettono di ridurre sensibilmente i tempi di esecuzione e i costi, mantenendo al contempo le garanzie di sicurezza derivanti dal collegamento con il Layer 1. In questo ambito, Polygon rappresenta una delle piattaforme più mature e ampiamente adottate, pienamente compatibile con l'ecosistema Ethereum. Dal punto di vista prestazionale, Polygon presenta un tempo medio di blocco di circa 2,2 secondi [21], circa sette volte inferiore rispetto a quello di Ethereum. Come riportato nel grafico di *Fig. 45*, considerando un'attesa tipica di tre conferme (tx.await(3)), il tempo medio di finalizzazione di una transazione su Ethereum risulta pari a circa 45 secondi, contro i soli 6,6 secondi su Polygon. Va tuttavia sottolineato che il numero di conferme necessarie per ritenere una transazione definitivamente sicura è variabile, generalmente le dApps usano un minimo di 12 blocchi. Il numero di blocchi per considerare una transazione confermata rappresenta un iperparametro progettuale, che incide direttamente sulla latenza percepita e deve essere scelto in funzione del livello di affidabilità richiesto dall'applicazione. Dal punto di vista economico, il vantaggio offerto da Polygon è evidente. A parità di gas consumato, il costo finale di una transazione dipende dal prezzo unitario del token nativo. Con ETH a 3.500 €

(20 agosto 2025) e MATIC a 0,32 €, operazioni che su Ethereum hanno un costo compreso tra 2 € e 17 € si riducono a pochi millesimi di euro su Polygon. Come mostrato dal grafico in *Fig. 46*, l'esecuzione di operazioni complesse quali split e merge, che su Ethereum risultano particolarmente onerose, diventa trascurabile in termini di costo sulla rete Polygon. Tale differenza rende sostenibile l'esecuzione frequente di operazioni articolate, aprendo la strada a scenari industriali caratterizzati da un'elevata interattività con la blockchain.

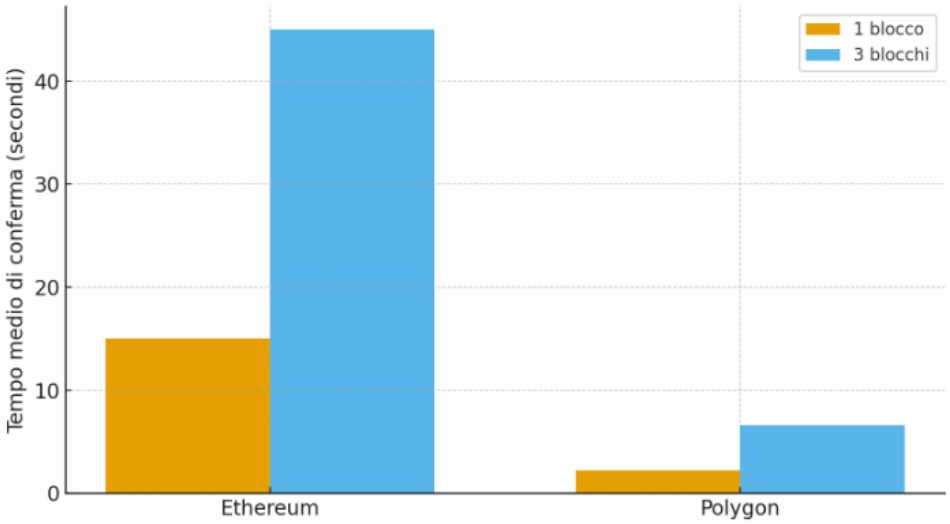


Figura 45: Tx.await(i) confronto Ethereum / Polygon

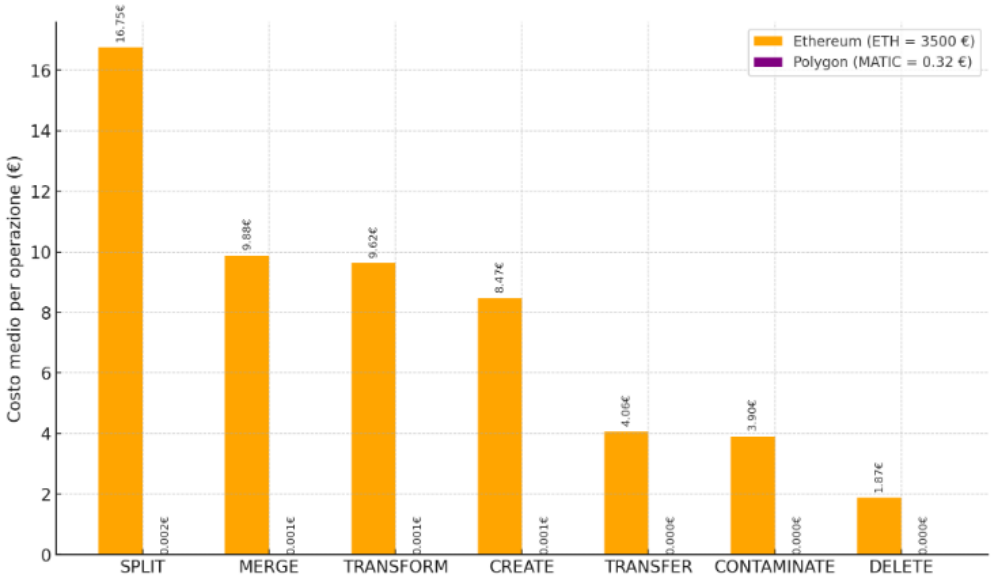


Figura 46: Confronto costi operazioni tra Ethereum e Polygon

Gestione delle chiavi utente

Il progetto sviluppato si è concentrato sulla logica core del sistema, fondamentalmente realizzando una PoC[16] comprendente la modellizzazione dei dati trattati, la trasposizione su blockchain, l'architettura e il monitoraggio con comparazione dei costi e delle prestazioni, senza includere un modulo di autenticazione. Nella configurazione attuale, l'identificazione degli attori avviene tramite *l'inserimento dell'**address** all'interno della richiesta HTTP*, mentre il backend utilizza un'utility che firma le transazioni per conto dell'utente. Questa soluzione, sebbene funzionale in fase di sviluppo e testing, *non è adottabile in un ambiente di produzione*, poiché non prevede alcun meccanismo di sicurezza: chiunque potrebbe dichiarare un indirizzo arbitrario e impersonare un altro utente. In un contesto reale, la gestione delle chiavi crittografiche rappresenta un elemento centrale per garantire sicurezza e affidabilità. Le principali strategie adottate nel panorama blockchain possono essere distinte in due modelli:

- **Custodial**: le chiavi private sono conservate da un'entità terza, ad esempio un exchange o un provider centralizzato. L'utente interagisce attraverso meccanismi di autenticazione tradizionali (username, password, autenticazione a due fattori), mentre la firma delle transazioni è delegata al custode. Questo modello semplifica l'esperienza utente e consente il recupero in caso di smarrimento, ma introduce dipendenza dal servizio e un rischio concentrato in caso di compromissione.
- **Non-custodial**: le chiavi rimangono sotto il controllo esclusivo dell'utente, tipicamente tramite wallet software come MetaMask o wallet hardware come Ledger e Trezor. In questo caso ogni transazione deve essere firmata direttamente dall'utente. Il vantaggio è la piena sovranità sulle proprie chiavi, ma a fronte di una maggiore responsabilità: la perdita della chiave privata o della seed phrase comporta la perdita definitiva dell'accesso.

Le differenze tra i due approcci sono sintetizzate nella *Fig. 47*, che mette a confronto i modelli *custodial* e *self-custody* evidenziandone i principali aspetti. Per un sistema come quello descritto, in cui le API sono utilizzate da attori eterogenei della supply chain, sarà necessario valutare quale modello adottare, bilanciando semplicità di utilizzo, livello di sicurezza e responsabilità attribuite agli utenti finali.

Caratteristica	Custodial	Self-custody
Conservazione chiavi	Presso un provider esterno	In locale, sotto il controllo dell'utente
Esperienza utente	Più semplice, simile a servizi web tradizionali	Richiede l'uso di un wallet e la gestione della seed
Sicurezza	Dipende dal custode, rischio di attacco centralizzato	Elevata, se l'utente protegge correttamente le chiavi
Recuperabilità	Possibile tramite il provider	Impossibile in caso di perdita delle chiavi
Responsabilità	In capo al servizio di custody	Completamente in capo all'utente
Esempi	Exchange centralizzati, provider SaaS	MetaMask, Ledger, Trezor, Trust Wallet

Figura 47: Confronto modelli Custodial e Self-custody

[16] *Proof of Concept*

Generazione degli ID

Attualmente gli identificativi dei lotti sono gestiti tramite un contatore globale incrementato a ogni nuova creazione. Questa scelta garantisce compatibilità con lo standard ERC-1155, che richiede ID numerici, e assicura unicità attraverso il consenso della blockchain. Tuttavia, tale approccio presenta alcune criticità. In primo luogo, la **concorrenza**: tutte le transazioni che creano nuovi lotti devono aggiornare la stessa variabile di stato, con conseguente serializzazione e possibile aumento dei tempi di attesa in caso di traffico elevato. In secondo luogo, la latenza dovuta all'esecuzione on-chain: la generazione dell'ID avviene tramite una transazione sulla blockchain e richiede il completamento del processo di consenso e la successiva inclusione della transazione in un blocco. Su Ethereum, questo implica un tempo minimo di attesa pari almeno alla produzione di un blocco (≈15 secondi) e, per garantire l'irreversibilità dell'operazione, l'attesa di un certo numero di blocchi di conferma che riducano il rischio di *riorganizzazioni della catena*[17]. Tale variabilità rende il tempo di creazione di un nuovo identificativo un fattore critico per le prestazioni complessive. Per ridurre l'impatto di queste limitazioni si possono considerare diverse possibilità, (riassunte in *Fig.48*):

- **Pre-allocazione di blocchi di ID**: il contratto può riservare in un'unica transazione un intervallo di ID consecutivi per un determinato attore. Questi ID possono poi essere utilizzati localmente senza richiedere ulteriori interazioni con la blockchain fino all'esaurimento dell'insieme.
- **Schemi di partizionamento**: invece di un contatore globale, gli ID possono essere generati su più spazi numerici separati, riducendo la contesa sulla stessa variabile e distribuendo il carico tra più contatori.
- **Generatori off-chain con validazione on-chain**: un attore può proporre un ID numerico generato a livello applicativo e il contratto verifica soltanto che non sia già stato utilizzato. In questo caso l'unicità è comunque garantita dal controllo on-chain, mentre la logica di generazione è demandata al livello applicativo. Grazie all'ampiezza dello spazio uint256, la probabilità di collisione risulta trascurabile, ma il contratto deve comunque respingere eventuali ID già occupati.

Queste soluzioni consentono di mantenere la compatibilità con ERC-1155 mitigando i limiti di scalabilità del modello puramente sequenziale. La scelta dipende dal bilanciamento desiderato tra semplicità, riduzione della latenza e complessità implementativa, considerando che ogni approccio porta con sé compromessi.

[17] si intende una situazione in cui la blockchain sostituisce temporaneamente alcuni blocchi recenti con un'altra versione della catena considerata più lunga o con maggiore *"difficulty"* cumulata. Questo può accadere, ad esempio, quando due blocchi vengono trovati quasi simultaneamente. Durante la riorganizzazione, le transazioni non ancora confermate possono essere annullate o riposizionate.

Approccio	Descrizione	Pro	Contro
Sequenziale on-chain	Incremento di `_lastTokenId` a ogni nuova creazione.	Semplice, sicuro, compatibile ERC-1155	Collo di bottiglia, latenza 1-12 blocchi
Pre-allocazione blocchi	Riservati intervalli di ID in un'unica transazione.	Riduce interazioni on-chain, più rapido	Gestione complessa, rischio spreco ID
Partizionamento range	Range separati per attori/categorie.	Parallelismo, meno contesa	Rigidità, serve schema iniziale
Off-chain randomico + validazione	ID generati off-chain (es. randomness crittografica), verificati dal contratto.	Scalabile, latenza ridotta, basso rischio collisione	Complessità maggiore, necessità di fallback se ID già usato

Figura 48: Confronto tra possibili soluzioni per la generazione degli ID

Multisignature e non-repudiation

Una possibile estensione riguarda il meccanismo di trasferimento dei token. Nella sua formulazione standard, l'operazione safeTransferFrom prevista dallo standard ERC-1155 richiede esclusivamente l'iniziativa e la firma crittografica del mittente, mentre il destinatario è un soggetto passivo che riceve il token senza dover esplicitare alcun consenso formale. Questa modalità, sebbene sufficiente nella maggior parte dei casi, non offre garanzie di **non ripudiabilità** da parte del destinatario, né formalizza l'acquisizione come azione concordata da entrambe le parti.

Un'evoluzione del modello potrebbe consistere nell'introdurre un meccanismo di **firma congiunta** sul transfer, mutuando concetti già noti nelle logiche di multisignature ed escrow [22]. In tale schema l'operazione di trasferimento verrebbe eseguita solo qualora **entrambi** gli attori coinvolti, cedente e acquirente, abbiano firmato digitalmente la stessa richiesta di cessione/acquisizione. In termini pratici, il flusso si articolerebbe in due passaggi:

1. Il mittente genera una proposta di trasferimento contenente i parametri essenziali (from, to, tokenId, quantità, eventuali metadati) e la firma con la propria chiave privata.
2. Il destinatario riceve la proposta, ne verifica la correttezza e aggiunge a sua volta la propria firma digitale.

Solo a questo punto la transazione, arricchita da entrambe le firme, viene inviata allo smart contract, che procede con la verifica crittografica e l'esecuzione del safeTransferFrom.

Questo approccio garantisce che ogni operazione di trasferimento rifletta un consenso esplicito da parte di entrambi gli attori, rendendo l'operazione **non ripudiabile per entrambi** e riducendo il rischio di contestazioni. La sua adozione richiederebbe un aumento della complessità del protocollo, in quanto le firme devono essere raccolte e gestite prima di essere validate on-chain, ma rafforza significativamente la robustezza del modello in scenari in cui la fiducia reciproca tra attori della supply chain non può essere assunta come garantita [22].

Conclusioni

Sintesi dei risultati

Obiettivi e contesto

Il lavoro di tesi ha perseguito l'obiettivo di realizzare un sistema di tracciabilità per supply chain basato su blockchain, sviluppato seguendo un percorso metodologico rigoroso che ha condotto dalla modellizzazione teorica alla dimostrazione pratica di fattibilità. Il punto di partenza è stato l'analisi delle supply chain moderne e l'individuazione delle loro esigenze fondamentali, inquadrate nel contesto normativo europeo del Digital Product Passport che introduce l'obbligo di un'identità digitale per i prodotti lungo l'intera filiera. Per fornire una base concettuale solida, è stata condotta un'analisi del modello Cerere sviluppato da Cimino et al. un quadro teorico consolidato in grado di descrivere con rigore i concetti fondamentali di attori, lotti e operazioni. Il modello formalizza le entità tracciabili attraverso costrutti essenziali e definisce sei schemi di attività fondamentali: *acquisizione, cessione, divisione, integrazione, alterazione e trasformazione* che catturano le operazioni principali delle supply chain. Questo modello teorico ha fornito il quadro concettuale necessario da cui partire per la traduzione in termini informatici, assicurando che l'implementazione mantenesse coerenza semantica con i processi reali di supply chain.

Progettazione e tokenizzazione ERC-1155

La selezione dello standard ERC-1155 è emersa da un'analisi comparativa degli standard ERC disponibili, rivelando come questa soluzione consenta di identificare univocamente i lotti con un token ID numerico e di rappresentare gli attori attraverso indirizzi blockchain. La scelta si è rivelata particolarmente adeguata per le sue caratteristiche: permette la gestione unificata di asset eterogenei in un singolo contratto, supporta operazioni in batch che riducono significativamente costi e latenza, e separa efficacemente le evidenze on-chain dai metadati off-chain tramite URI associati ai token id specifici. Il processo di mappatura del modello teorico sulle primitive ERC-1155 ha consentito di tradurre efficacemente le operazioni astratte del modello Cerere sulle primitive fondamentali dello standard ERC1155 implementato tramite le librerie OpenZeppelin. Le operazioni astratte sono state mappate rispettivamente su mint, safeTransferFrom, combinazioni di burn/mint per split e merge, e burn per la cancellazione, ottenendo un parallelismo diretto tra processi di filiera e operazioni token-based e introducendo minime variazioni rispetto al modello originario, per esempio, il trattamento delle operazioni di acquisition e providing come unica operazione atomica di trasferimento di token tra attori.

Architettura e modello dei dati

Su queste basi è stato progettato un modello dei dati aggregato che integra due domini informativi distinti: PublicData, contenente metadati descrittivi statici destinati alla consultazione e TraceabilityData, che raccoglie informazioni dinamiche di stato, prove blockchain e cronologia delle operazioni. Questo modello permette di mantenere coerenza tra la rappresentazione logica del lotto e la sua implementazione digitale, garantendo che ogni lotto mantenga un'identità univoca, una scheda informativa immutabile e una cronologia operativa interrogabile.

Da un punto di vista architetturale le informazioni sono state divise tra diversi layer di persistenza per rispettare i requisiti specifici della supply chain. L'architettura a tre livelli sviluppata integra domini tecnologici complementari: la blockchain Ganache/Ethereum registra le operazioni tokenizzate garantendo immutabilità, trasparenza e non-ripudiabilità attraverso transazioni di token ERC-1155 ed eventi; IPFS fornisce storage distribuito content-addressed dei metadati e allegati, garantendo che ogni documento sia identificabile univocamente dal suo hash crittografico e verificabile indipendentemente dal punto di pubblicazione; MongoDB opera come livello applicativo per query ad alte prestazioni, aggregando dati on-chain, metadati IPFS e informazioni di business per consentire visualizzazioni efficienti. Questa separazione delle responsabilità ha permesso di coniugare efficacemente i requisiti di trasparenza propri delle reti distribuite con l'efficienza operativa necessaria nelle applicazioni industriali.

API RESTful e validazione del sistema

Per rendere il sistema accessibile e utilizzabile da applicazioni esterne è stata progettata e implementata un'API RESTful basata su Node.js e Express. L'interfaccia espone endpoint semanticamente coerenti che seguono rigorosamente i principi RESTful, utilizzando URI intuitivi che rendono immediatamente comprensibile la risorsa e l'azione richiesta. L'API orchestra le interazioni tra i tre livelli di storage, mascherando efficacemente la complessità dell'architettura distribuita agli utenti finali e fornendo operazioni di alto livello per la gestione completa del ciclo di vita dei lotti. Gli endpoint sono stati progettati secondo una logica che distingue operazioni collection level, che agiscono su insiemi di lotti con riferimenti nel payload, e resource level, che operano su singoli lotti identificati nel path, permettendo di gestire flessibilmente sia trasformazioni complesse multi-lotto che interazioni elementari.

La validazione del sistema è stata condotta attraverso una suite di test articolata su più livelli complementari. I test unitari hanno verificato il corretto funzionamento di ogni singolo endpoint, assicurando comportamenti conformi alle specifiche e coprendo tanto i casi di successo quanto gli scenari di errore. I test end-to-end hanno simulato flussi completi di supply chain concatenando multiple operazioni in sequenza, verificando che il sistema mantenesse consistenza globale lungo l'intero percorso operativo. Test specifici sui flussi di filiera hanno dimostrato la corretta gestione di scenari complessi con convergenze e diramazioni tipiche delle supply chain reali, confermando la correttezza funzionale del sistema e la consistenza dei dati tra i diversi livelli di storage.

Analisi delle prestazioni e risultati

L'analisi prestazionale condotta attraverso benchmark dedicati ha fornito metriche quantitative fondamentali per valutare l'efficienza operativa. In ambiente simulato locale, le operazioni si distribuiscono chiaramente in tre categorie di performance: operazioni complesse che richiedono 2-3,5 secondi per la combinazione di transazioni on-chain, caricamenti IPFS e aggiornamenti database; operazioni intermedie con tempi attorno a 1,3 secondi; operazioni leggere che si completano in meno di un secondo. L'analisi ha identificato chiaramente la blockchain come principale fattore di latenza, mentre IPFS e MongoDB contribuiscono marginalmente ai tempi totali. Parallelamente, l'analisi economica *(su Ethereum mainnet simulata)* ha rivelato tre fasce di costo distinte: operazioni onerose per CREATE, MERGE, TRANSFORM; costi medi per TRANSFER e CONTAMINATE; operazioni economiche per BURN/DELETE.

Il progetto dimostra la fattibilità del sistema, costituendo una proof of concept riuscita che conferma aspetti critici fondamentali: la possibilità concreta di mappare un modello teorico consolidato di supply chain su un'architettura token-based mantenendo coerenza semantica; l'efficacia dell'integrazione tra blockchain, storage distribuito e database; la scalabilità dell'approccio ERC-1155 per la gestione di lotti con quantità variabili e metadati complessi; la robustezza del sistema di tracciabilità end-to-end attraverso multiple trasformazioni e passaggi di proprietà. I risultati posizionano il prototipo come contributo originale allineato con le recenti direttive europee sul Digital Product Passport, dimostrando che le tecnologie blockchain possono effettivamente supportare i requisiti di trasparenza e verificabilità richiesti dalla normativa europea. Il codice sorgente sviluppato a supporto della presente tesi è disponibile su GitHub [23].

Limiti del modello

Condizioni sperimentali e prestazioni

L'esperienza di sviluppo e la fase di testing hanno permesso di identificare una serie di criticità rispetto ad una eventuale transizione verso ambienti produttivi. Le misurazioni sono state effettuate esclusivamente in ambiente locale, con una blockchain simulata istanziata tramite Ganache dove la generazione dei blocchi è pressoché istantanea e la validità delle transazioni è stata riconosciuta dopo un solo blocco. Queste condizioni determinano latenze ottimistiche che non riflettono le reali condizioni operative di una rete pubblica distribuita, dove i tempi di propagazione e i meccanismi di consenso introducono ritardi significativamente maggiori. I tempi di blocco rappresentano un fattore critico che incide direttamente sulla reattività del sistema. In un contesto reale su Ethereum mainnet, considerando un tempo medio di blocco di circa 15 secondi e la necessità di attendere multiple conferme per garantire sicurezza, le operazioni complesse potrebbero richiedere da un minimo di 45 secondi a diversi minuti, con impatti significativi sull'esperienza utente.

I costi di esecuzione su Ethereum mainnet rappresentano una limitazione economica che rende proibitivo l'uso intensivo del sistema. Le transazioni comportano costi elevati che variano in base alla complessità dell'operazione e alle condizioni di congestione della rete, limitando severamente la scalabilità economica del modello. Anche adottando strategie di ottimizzazione come le operazioni in batch, i costi rimangono incompatibili con applicazioni industriali, rendendo necessaria l'esplorazione di alternative come blockchain di Layer-2.

Limitazioni architetturali

La gestione dell'autenticazione è stata deliberatamente semplificata per concentrare l'attenzione sulla proof of concept. Attualmente le firme delle transazioni sono gestite dal backend utilizzando chiavi configurate nel file di configurazione, che opera per conto dell'utente. Questa soluzione presenta vulnerabilità significative che la rendono inadeguata per scenari produttivi: manca un sistema di gestione delle chiavi utente. In un ambiente di produzione, la gestione delle chiavi andrebbe delegata agli utenti attraverso wallet dedicati oppure andrebbe sviluppato o integrato un sistema custodial di gestione delle chiavi per loro conto.

La generazione degli identificativi dei lotti è basata su contatore globale quindi costituisce un collo di bottiglia architetturale che limita la scalabilità. Questa logica introduce potenziali conflitti nella creazione dei blocchi Ethereum in scenari ad alto throughput: ogni aggiornamento degli ID rappresenta una transazione che modifica lo stato del contratto, quindi quando diverse transazioni concorrenti sono in attesa di essere confermate durante il processo di consenso, si possono produrre conflitti e alcune transazioni possono essere scartate, causando fallimenti e necessità di re-invio. Questo meccanismo può generare ritardi aggiuntivi e costi imprevisti in situazioni di alta concorrenza.

La modellazione delle trasformazioni è stata semplificata implementando trasformazioni *n a 1*, mentre per rappresentare più fedelmente la complessità dei processi industriali reali sarebbe necessario supportare trasformazioni *n a m* che permettano di riprodurre scenari con output multipli eterogenei. Questa semplificazione limita l'applicabilità del sistema a processi produttivi complessi.

Prospettive di ricerca

Layer-2 e ottimizzazioni

Le prospettive di sviluppo futuro si declinano in più direttrici tra loro complementari, volte a superare le criticità emerse e a delineare un percorso di evoluzione verso un sistema industrialmente sostenibile. La migrazione verso blockchain di livello 2, come Polygon, rappresenta la priorità strategica più critica per superare simultaneamente le limitazioni prestazionali ed economiche. L'analisi comparativa condotta ha mostrato riduzioni drastiche sia nei tempi, da circa 45 secondi a circa 6,6 secondi per tre conferme, che nei costi, da euro a frazioni di centesimo, rendendo sostenibile l'esecuzione frequente di operazioni complesse. Il sistema beneficerebbe significativamente del miglioramento della gestione delle operazioni in batch, già implementate nel prototipo ma suscettibili di ottimizzazioni mirate. Lo sviluppo di criteri ottimizzati per l'aggregazione delle transazioni potrebbe permettere la gestione contemporanea di grandi quantità di lotti, ammortizzando cioè il tempo necessario per la conferma di una transazione on chain su un ampio numero di lotti. Il miglioramento della gestione degli ID attraverso strategie alternative al contatore globale rappresenta un'area fondamentale: soluzioni basate su generazione off-chain validata on-chain potrebbero sfruttare l'ampiezza dello spazio uint256 per minimizzare collisioni, mentre approcci di pre-allocazione di range o partizionamento potrebbero distribuire il carico riducendo la contesa e migliorando le performance.

Estensioni funzionali e cross-chain

L'integrazione di moduli di gestione delle chiavi rappresenta un'evoluzione assolutamente necessaria: Esistono due alternative principali: sistemi self-custody, come MetaMask, che garantiscono sovranità sulle chiavi ma richiedono maggiore responsabilità da parte degli utenti, e modelli custodial che semplificano l'esperienza ma introducono dipendenza da terze parti. L'introduzione di sistemi di gestione delle chiavi dovrà bilanciare sicurezza e usabilità per diversi profili di utenti industriali, considerando le specifiche esigenze di settori con differenti livelli di competenza tecnica e requisiti di sicurezza.

L'implementazione di trasformazioni *n a m* rappresenta un altro aspetto di rilevo per supportare la reale complessità dei processi industriali. Il sistema dovrebbe supportare operazioni che da multipli input generano multipli output eterogenei, catturando scenari più complessi, corrispondenti alla realtà industriale.

Una possibilità particolarmente interessante riguarda l'estensione cross-chain del sistema. Avendo progettato un'architettura basata su API REST, si aprono opportunità per lo sviluppo di bridge di compatibilità con blockchain eterogenee che supportino standard analoghi a ERC-1155. Il meccanismo potrebbe basarsi su bridge il cui funzionamento consiste fondamentalmente nel blocco di token su una chain e nella generazione di token su un’altra chain, permettendo la gestione di supply chain che adottano blockchain eterogenee. Questo approccio potrebbe essere particolarmente vantaggioso per supply chain che coinvolgono regioni con preferenze tecnologiche differenti o requisiti normativi specifici.

Validazione su reti pubbliche

La validazione su reti pubbliche e Layer-2 costituisce il passaggio fondamentale per confermare la scalabilità reale oltre l'ambiente simulato. Test su reti pubbliche con configurazioni di conferma variabili e benchmark comparativi sotto carico elevato forniranno metriche definitive per l'ottimizzazione delle performance e la definizione di parametri operativi ottimali. Andrebbero considerate diverse condizioni di rete, picchi di traffico, e scenari di stress per validare la robustezza del sistema in condizioni operative reali.

Il progetto di tesi sviluppato dimostra quindi come un modello teorico consolidato di supply chain possa essere mappato efficacemente su uno standard token-based, identificando ERC-1155 come standard pienamente adatto e definendo un'architettura robusta verificata attraverso test sistematici. Le criticità emerse costituiscono il punto di partenza per futuri sviluppi del sistema, mentre le prospettive delineate forniscono una roadmap concreta per trasformare la proof of concept in un sistema scalabile e interoperabile. Questo percorso evolutivo è orientato a supportare scenari industriali reali e a contribuire alla realizzazione dell'ecosistema del Digital Product Passport europeo, dimostrando che le tecnologie blockchain possano effettivamente soddisfare i requisiti delle moderne supply chain.

Bibliografia

1] E. P. a. Council, *Regulation (EC) No. 178/2002 of the European Parliament and of the Council of 28 January 2002, laying down the general principles and requirements of food law, Chapter V.,* Brussels: Official Journal of the European Communities, 2002.

[2] E. P. a. Council, *Regulation (EU) 2024/1781 of the European Parliament and of the Council of 13 June 2024, establishing a framework for the setting of ecodesign requirements for sustainable products, OJ L 2024/1781.,* Brussels: Official Journal of the European Union, 2024.

[3] E. Commission, *Proposal for a Regulation of the European Parliament and of the Council on the establishment of the digital euro (COM(2023) 369 final), 28 June 2023,* Brussels: European Commission, 2023.

[4] D. Simchi-Levi, P. Kaminsky e E. Simchi-Levi, Designing and Managing the Supply Chain, New York, NY, USA: McGraw-Hill Education, 2002.

[5] M. Christopher, Logistics and Supply Chain Management, London, UK: Pearson, 2022.

[6] M. Perano, A. Cammarano, V. Varriale, C. Del Regno, F. Michelino e M. Caputo, «Embracing Supply Chain Digitalization and Unphysicalization to Enhance Supply Chain Performance: A Conceptual Framework,» *International Journal of Physical Distribution & Logistics Management,* vol. 53, n. 6, p. 628–659, 2023.

[7] X. Lu, «A Review of Supply Chain Digitalization and Emerging Perspectives,» *Machines,* vol. 13, n. 2, p. 47, 2025.

[8] M. Cimino, B. Lazzerini, F. Marcelloni e A. Tomasi, «Cerere: An information system supporting traceability in the food supply chain,» in *Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05)*, Munich, Germany, 2005.

[9] «EIP-20, "ERC-20 Token Standard.",» [Online]. Available: https://eips.ethereum.org/EIPS/eip-20.

[10] «Ethereum.org, "Token standards.",» 11 Giugno 2024. [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/. [Consultato il giorno 11 Agosto 2025].

[11] «EIP-721, "Non-Fungible Token Standard.",» [Online]. Available: https://eips.ethereum.org/EIPS/eip-721.

[12] «EIP-1155, "Multi Token Standard.",» [Online]. Available: https://eips.ethereum.org/EIPS/eip-1155.

[13] «Ethereum.org, “ERC-1155 Multi-Token Standard.”,» [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-1155/.

[14] «OpenZeppelin, “Contracts API – ERC-1155.”,» [Online]. Available: https://docs.openzeppelin.com/contracts/5.x/api/token/ERC1155.

[15] «OpenSea Docs, “Metadata standards (ERC-721 ed ERC-1155).”,» [Online]. Available: https://docs.opensea.io/docs/metadata-standards.

[16] «IPFS Docs, “Content Identifiers (CIDs) and content addressing.”,» [Online]. Available: https://docs.ipfs.tech/concepts/content-addressing/.

[17] «Circle, “USDC on Ethereum.”,» [Online]. Available: https://www.circle.com/multi-chain-usdc/ethereum.

[18] «MakerDAO, “Dai contract – detailed documentation.”,» [Online]. Available: https://docs.makerdao.com/smart-contract-modules/dai-module/dai-detailed-documentation.

[19] «OpenZeppelin, “Solidity Contracts – build secure smart contracts.”,» [Online]. Available: https://www.openzeppelin.com/solidity-contracts.

[20] «Ethereum.org, “Blocks.”,» [Online]. https://ethereum.org/developers/docs/blocks/.

[21] «Flipside Crypto, “Polygon Block Performance | mdburns.”,» [Online]. Available: https://flipsidecrypto.xyz/mdburns/polygon-block-performance-DF2VnS.

[22] «CSIRO / Data61, “Multiple Authorisation (aka., Multisignature).”,» [Online].

[23] M. Sica, “Tracker SupplyChain ERC1155,” GitHub repository, 2025. [Online]. Available: https://github.com/SMattii/Tracker_SupplyChain_ERC1155