Università della Calabria
Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Tesi di Laurea

Blockchain e IoT
tranciabilità e controlli automatici

Relatore:
Prof. Mario Alviano

Candidata:
Paola Guarasci
Matricola 231847

Anno Accademico 2023/2024

Indice

Sommario

Lorem ipsum...

1 Introduzione

2 Background

2.1 Digital Twin

2.2 Blockchain

La tecnologia dei registri distribuiti, distribuited ledger tecnlogy o DLT, consenta di utilizzare dati digitali in modo distribuito, condiviso e sincronizzato. Non necessita di una entità centrale. La blockchain identifica un particolare tipo di registro distribuito.

Scalabilità

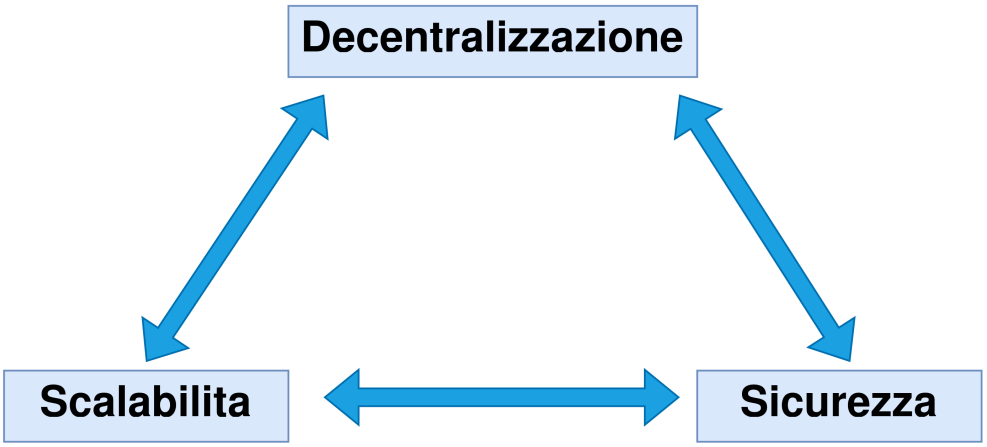

Figura 1: Il trilemma della blockchain

Il trilemma delle blockchain

Layer1 e Layer2 Le reti blockchain principali hanno definite *layer 1* e rappresentalo reti che funzionano da sole. Le reti layer 2, di contro, sono reti che si pongo, utilizzando diverse

metodologie, come sovrastuttura rispetto alla layer 1 che espandono. Di fatto rappresentano appunto delle espansioni delle reti principali. Il motibvo

Algoritmi di consenso

Permiossionless e Permissioned

Ethereum

Polygon

Tools

2.3 ASP

2.4 Concetti di Tracciabilità e di Filiera

3 Analisi dello stato dell'arte della tecnologia blockchain applicata alle supplychain

4 Definizione di un caso d'uso

Il caso d'uso che si vuole analizzare riguarda la tracciabilità di un prodotto inserito nella filiera agroalimentare. Un esempio potrebbe essere la tracciabilità di un ipotetico prodotto surgelato, ad esempio un minestrone, di cui si vuole tenere traccia, in modo immutabile dalla raccolta delle materie prime fino alla vendita finale.

4.1 Progettazione

Il software presenta un'architettura client-server con un componente principale, il server, ed un componente secondario, il client. Il client è un client non corrisponde del tutto alla definizione di client stupido. Si avvicina infatti più alla definizione di fat client. Il sistema implementa un'architettura a tre livelli in cui il livello di presentazione è stratificato. I livelli di un'applicazione a tre tier sono:

- Livello di presentazione
- Livello applicazione (business logic)

- Livello dati

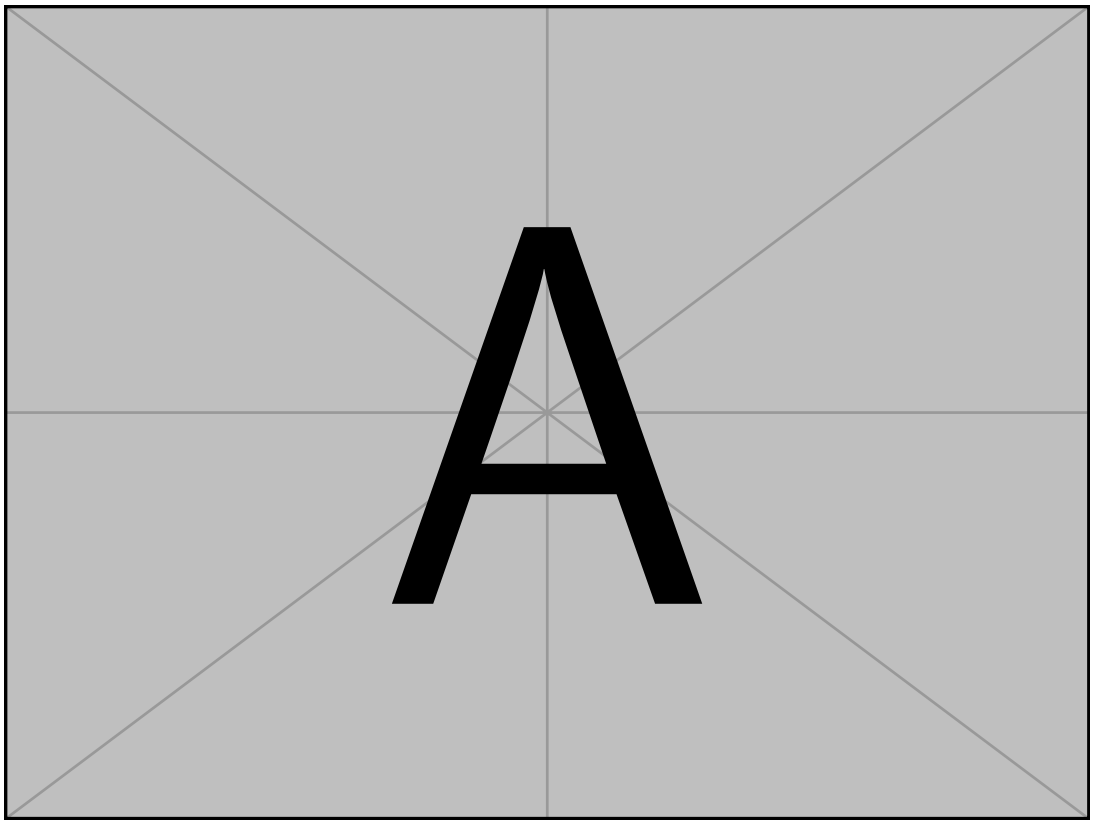

Figura 2: Architettura

Si è deciso di utilizzare una struttura monolitica e non a microservizi.La comunicazione tra componente server e componente client avviene tramite RestAPI. La costruzione di api rest implica alcuni vincoli architeturali [6]

- Bisogna definire un'interfaccia uniforme - Avere un'architettura client server in cui un componente chiede risorse (client) e un altro le espone (server) - Il server è senza stato ovvero il server non ha memoria delle connessioni precedenti - Per le richieste in lettura si può prevedere un meccanismo di cache in cui il server propone gli stessi contenuti. Bisogna prevedere anche criteri di aggiornamento e invalidamento dei contenuti in cache. - Implementare un sistema a livelli, in cui il client si connette ad un servizio senza conoscere l'architettura interna - Codice on demand (opzionale) quando necessario, oltre alle risorse statiche, è possibile inserire del codice eseguibile nella risposta del server

L'autenticazione utilizza il protocollo OAuth2 ed un'istanza di Keycloak come authentication server. [7]

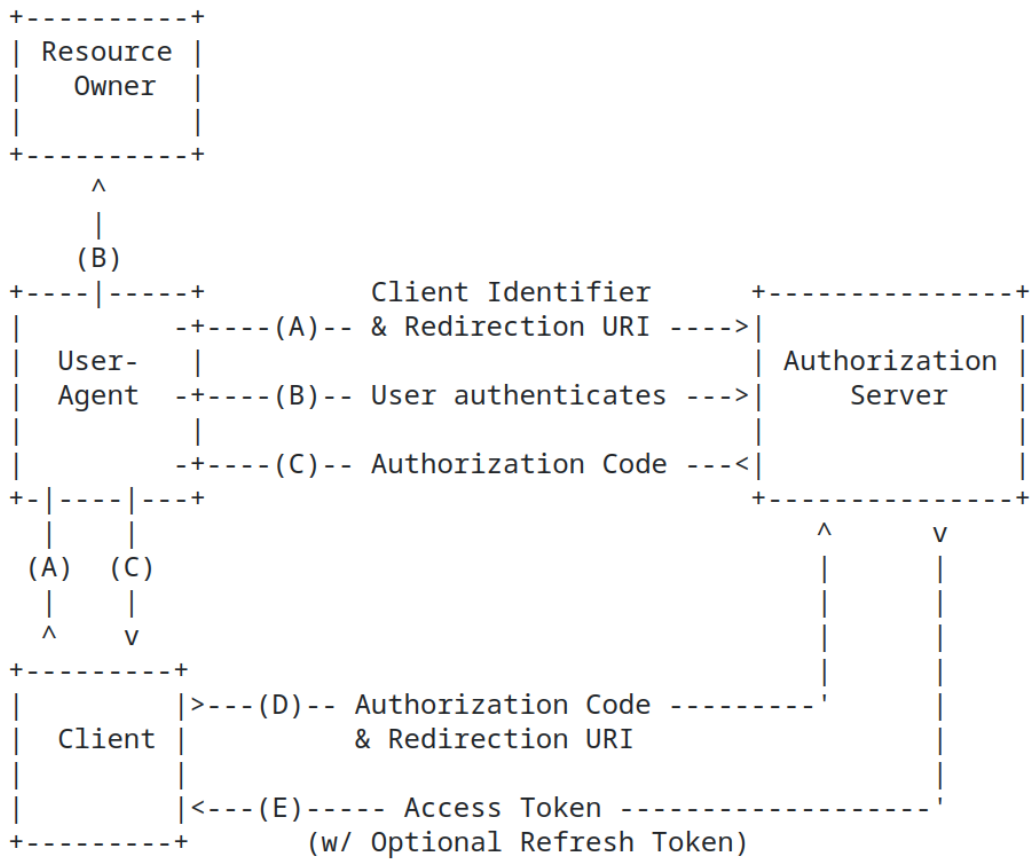

Figura 3: Authorization Flow in OAuth2 [7]

Lo stack tecnologico utilizzato comprende Sono stati presi in esame diversi framework per la componente server. In ambito NodeJS sono stati presi in considerazione NestJS ed Express.

Descrizione di Nest Nest (NestJS) is a framework for building efficient, scalable Node.js server-side applications. It uses progressive JavaScript, is built with and fully supports TypeScript (yet still enables developers to code in pure JavaScript) and combines elements of OOP (Object Oriented Programming), FP (Functional Programming), and FRP (Functional Reactive Programming). [...] In recent years, thanks to Node.js, JavaScript has become the ”lingua franca” of the web for both front and backend applications. This has given rise to awesome projects like Angular, React and Vue, which improve developer productivity and enable the creation of fast, testable, and extensible frontend applications. However, while plenty of superb libraries, helpers, and tools exist for Node (and server-side JavaScript), none of them effectively solve the main problem of - Architecture. Nest provides an out-of-the-box

application architecture which allows developers and teams to create highly testable, scalable, loosely coupled, and easily maintainable applications. The architecture is heavily inspired by Angular. [3]

Descrizione di Express Express è un framework per applicazioni web Node.js flessibile e leggero che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili. Con una miriade di metodi di utilità HTTP e middleware a disposizione, la creazione di un'API affidabile è un processo facile e veloce. Express fornisce uno strato sottile di funzionalità di base per le applicazioni web, senza nascondere le funzioni Node.js. [4]

Descrizione di Spring e Spring Boot Spring è un framework java. Spring Boot è una sua estensione. Tipicamente si utilizzano insieme, è ormai difficile immaginare di usare Spring senza Spring Boot. Spring si basa sull'inversione del controllo e sull'injection delle dipendenze. Il principio di design di inversione del controllo, IoC, descrive quelle situazioni in cui un componente software applicativo riceve il controllo da parte di un componente di libreria e non il contrario come avviene in una programmazione procedurare tradizionale in cui il software applicativo utilizza (e controlla) le componenti di libreria. In questo contesto si inserisce l'iniezione delle dipendenze che è un altro aspetto fondamentale del framework Spring, che elimina dall'applicazione ogni logica di inizializzazione. È il framework stesso che quando necessario inietta un riferimento all'implementazione del servizio richiesto. [10]

Spring Boot crea applicazioni Spring standalone, con un application server embedded a scelta tra Tomcat o Jetty. Spring Boot semplifica il processo di sviluppo perché abbraccia lo stile Convention-over-configuration ed utilizza configurazioni standard la dove il programmatore non ha definito una configurazione personalizzata per quel particolare aspetto del software in svilupo. Esistono difatti moduli di spring boot che fanno da "starter" per i diversi moduli di spring. La configurazione, quando è necessaria, non deve in ogni caso essere redatta in XML. [16]

Scelta del framework e considerazioni personali Express.JS si è rivelato eccessivamente granulare e di basso livello per gli scopi del progetto. NestJS e Spring, sebbene molto simili tra di loro, soprattutto su alcuni aspetti di design quali l'inversione del controllo e l'injection delle dipendenze, che sono concetti che si possono ritrovare in entrambi i framework, utilizzano due ecosistemi differenti, il primo NodeJS ed il secondo Java. Si è scelto di utilizzare Java per poter utilizzare a pieno i vantaggi di una programmazione orientata agli oggetti in un ambiente multithread. NodeJs ha una natura asincrona e dalla documentazione di node.js [12] si legge

In a one-thread-per-client system like Apache, each pending client is assigned its own thread. If a thread handling one client blocks, the operating system will interrupt it and give another client a turn. The operating system thus ensures that clients that require a small amount of work are not penalized by clients that require more work. Because Node.js handles many clients with few threads, if a thread blocks handling one client's request, then pending client requests may not get a turn until the thread finishes its callback or task. The fair treatment of clients is thus the responsibility of your application. This means that you shouldn't do too much work for any client in any single callback or task. This is part of why Node.js can scale well, but it also means that you are responsible for ensuring fair scheduling. The next sections talk about how to ensure fair scheduling for the Event Loop and for the Worker Pool.

Il server utilizza il pattern repository ed altri pattern in maniera più o meno esplicita

Gestione delle transazioni

LATO AUTENTICAZIONE

LATO CLIENT

I framework lato client presi in esame sono Vue.js, Angular e React. I dati di diffusione di questi framework li pongono ai primi tre posti di una ipotetica classifica basata sull'utilizzo misurato in numero di repository pubblici. Se si guarda all'utilizzo al primo posto troviamo React seguito da Vue e poi da Angular. Riguardo l'awareness (conoscenza) invece Angular passsa in prima posizione, Vue mantiene il secondo posto e React scivola in terza posizione [17].

Vue.js

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative, component-based programming model that helps you efficiently develop user interfaces of any complexity. [...] Vue is a framework and ecosystem that covers most of the common features needed in frontend development. But the web is extremely diverse - the things we build on the web may vary drastically in form and scale. With that in mind, Vue is designed to be flexible and incrementally adoptable. [...] Despite the flexibility, the core knowledge about how Vue works is shared across all these use cases. Even if you are just a beginner now, the knowledge gained along the way will stay useful as you grow to tackle more ambitious goals in the future. If you are a veteran, you can pick the optimal way to leverage Vue based on the problems you are trying to solve, while retaining the same productivity. This is why we call Vue "The Progressive Framework": it's a framework that can grow with you and adapt to your needs. [20]

React

React (noto anche come React.js o ReactJS) è una libreria open-source, front-end, JavaScript per la creazione di interfacce utente. È mantenuto da Meta (già Facebook) e da una

comunità di singoli sviluppatori e aziende. React può essere utilizzato come base nello sviluppo di applicazioni a pagina singola ma è utilizzabile anche su mobile tramite React Native, una libreria sempre sviluppata da Meta che tramuta i componenti React in componenti nativi (iOS e Android). Tuttavia, React si occupa solo del rendering dei dati sul DOM, pertanto la creazione di applicazioni React richiede generalmente l'uso di librerie aggiuntive per lo state management e il routing. Redux e React Router sono i rispettivi esempi di tali librerie. A questo fine è possibile utilizzare anche dei framework terzi, come ad esempio Next.js. [14]

Angular.JS

Angular is a web framework that empowers developers to build fast, reliable applications. Maintained by a dedicated team at Google, Angular provides a broad suite of tools, APIs, and libraries to simplify and streamline your development workflow. Angular gives you a solid platform on which to build fast, reliable applications that scale with both the size of your team and the size of your codebase. [1]

Scelta e motivazioni

4.1.1 Analisi dei requisiti

I requisiti funzionali di alto livello sono i seguenti:

- Ogni utente registrato opera per conto di una compagnia
- Un utente registrato deve poter creare tipologie di prodotto, indicando un template di ricetta e di processo produttivo
- Un utente registrato deve poter creare un nuovo lotto di prodotti di una determinata tipologia, con una ricetta, uno o più documenti allegati ed indicando le fasi di produzione di cui si vuole tenere traccia
- Un utente registrato deve poter trasferire (in modo atomico) un lotto di prodotti ad un'altra compagnia
- Un utente registrato deve poter inserire un documento e notarizzarlo
- Un utente registrato deve poter accedere in lettura ai dati di notarizzazione dei documenti caricati
- Un utente registrato deve poter trasferire con accettazione un lotto di prodotti ad un'altra compagnia
- Un utente registrato deve poter rifiutare il trasferimento con accettazione di un lotto trasferito alla compagnia per cui opera

- Un utente registrato deve poter accettare il trasferimento con accettazione di un lotto trasferito alla compagnia per cui opera
- Un utente registrato deve poter annullare il trasferimento con accettazione di un lotto disposto dalla sua compagnia
- Un utente registrato deve poter visualizzare le informazioni di tracciamento relative ai lotti di cui è in possesso la sua compagnia
- Un utente registrato deve poter avviare una spedizione per i trasferimenti conclusi e avviati dalla sua compagnia
- Un utente registrato deve poter visualizzare informazioni telemetriche delle spedizioni in corso
- Un utente non registrato deve poter visualizzare le informazioni pubbliche di tracciamento di un lotto di produzione

s I requisiti non funzionali sono applicabili alla quasi totalità dei sistemi, ovvero:

- Sicurezza: Il sistema deve essere protetto da accessi non autorizzati.
- Performance: Il sistema deve essere in grado di gestire il numero richiesto di utenti senza alcun degrado delle prestazioni.
- Scalabilità: Il sistema deve essere in grado di aumentare o diminuire in base alle esigenze.
- Disponibilità: Il sistema deve essere disponibile quando necessario.
- Manutenzione: Il sistema deve essere di facile manutenzione e aggiornamento.
- Portabilità: Il sistema deve essere in grado di funzionare su piattaforme diverse con modifiche minime.
- Affidabilità: Il sistema deve essere affidabile e soddisfare i requisiti dell'utente.
- Usabilità: Il sistema deve essere facile da usare e da capire.
- Compatibilità: Il sistema deve essere compatibile con altri sistemi.
- Compliance: Il sistema deve essere conforme a tutte le leggi e i regolamenti applicabili.

4.1.2 Casi d'uso

Sono stati individuati i seguenti casi d'uso:

- CU1: Registrazione di un tipo di prodotto
- CU2: Registrazione di un lotto prodotto
- CU3: Tracciamento di un lotto (utente registrato)
- CU4: Tracciamento di un lotto (utente non registrato)
- CU5: Trasferimento di un lotto (atomico) [INCLUDE trasferimento]
- CU6: Trasferimento di un lotto (con accettazione) [INCLUDE trasferimento]
- CU7: Accettazione di un trasferimento
- CU8: Ritiro di un trasferimento
- CU9: Rifiuto di un trasferimento
- CU10: Notarizzazione di un documento
- CU11: Avvio di una spedizione
- CU12: Registrazione di un utente
- CU13: Login di un utente

Specifiche dei casi d'uso:

Registrazione di un tipo di prodotto	
ID	CU1
Attori	Utente registrato
Precondizioni	Utente autenticato in piattaforma
Sequenza	• Il caso d'uso inizia quando l'utente clicca su "crea nuovo tipo di prodotto" • L'utente inserisce un nome • Se l'utente clicca su "aggiungi ricetta" – Il sistema avvia il caso d'uso CU1a • Se l'utente clicca su salva – il sistema salva il nuovo tipo di prodotto • Se l'utente clicca su cancella – il sistema annulla l'inserimento e non salva nessun dato
Post-condizioni	–

Tabella 1: Caso d'uso 1 - Registrazione di un tipo di prodotto

Creazione di una ricetta (estende CU1 Registrazione di un tipo di prodotto)	
ID	CU1a
Attori	Utente registrato
Precondizioni	Utente autenticato in piattaforma
Sequenza	• Il caso d'uso inizia quando l'utente clicca su "aggiungi ricetta" durante la creazione di un tipo di prodotto • L'utente seleziona un tipo di prodotto tra quelli già registrati e la quantità percentuale desiderata. • Il sistema memorizza temporaneamente i dati inseriti • Il sistema consente di inserire un nuovo item di ricetta • Il sistema consente di rimuovere ogni item inserito • Se l'utente clicca su aggiungi item – vai al passo 2 • Se l'utente clicca su rimuovi item – il sistema elimina l'item selezionatos – vai al passo 1 • Se l'utente clicca su cancella – il sistema annulla l'inserimento della ricetta, termina il caso d'uso e prosegue il caso d'uso CU1 • Se l'utente clicca su salva – il sistema salva temporaneamente la ricetta, termina il caso d'uso e prosegue il caso d'uso CU1
Post-condizioni	Se l'utente ha cliccato su salva la ricetta è stata salva correttamente nei dati temporanei del caso d'uso CU1

Tabella 2: Caso d'uso 1a - Creazione di una ricetta (estende CU1 Registrazione di un tipo di prodotto)

Registrazione di un lotto prodotto)	
ID	CU2
Attori	Utente registrato
Precondizioni	L'utente è autenticato in piattaforma e la compagnia per cui opera l'utente ha almeno un tipo di prodotto registrato.
Sequenza	• Il caso d'uso inizia quando l'utente clicca su "Inserisci nuovo lotto di prodotto" • L'utente inserisce identificativo, descrizione e quantità del lotto • L'utente seleziona il tipo di prodotto tra i tipi di prodotto già inseriti in piattaforma • Il sistema carica il processo produttivo del tipo di prodotto selezionato. • L'utente può selezionare uno o più step del processo produttivo del tipo selezionato • Se il tipo di prodotto ha una ricetta
Post-condizioni	• Il lotto di prodotto è stato creato • Le quantità delle materie prime sono state correttamente aggiornate • La creazione del lotto di prodotto è stata notarizzata • L'utente può visualizzare la notarizzazione nell'elenco notarizzazioni

Tabella 3: Caso d'uso 2 - Registrazione di un lotto di prodotto

Tracciamento di un lotto (utente registrato)	
ID	CU3
Attori	Utente registrato
Precondizioni	L'utente è autenticato in piattaforma e la compagnia per cui opera l'utente ha almeno un lotto di prodotto registrato.
Sequenza	• Il caso d'uso inizia quando l'utente clicca su "Traccia" di un lotto di produzione • Il sistema fornisce informazioni di tracciamento combinando la lista di materie prime, i singoli processi di ogni materia prima e le informazioni di notarizzazione con processi e notarizzazione del prodotto principale • Se l'utente seleziona Notarizza in uno qualunque degli step visibili inizia il caso d'uso CU3a5
Post-condizioni	–

Tabella 4: Caso d'uso 3 - Tracciamento di un lotto (utente registrato

Notarizzazione di uno step di processo (Estende: CU34)	
ID	CU3a
Attori	Utente registrato
Precondizioni	L'utente è autenticato in piattaforma, la compagnia per cui opera l'utente ha almeno un lotto di prodotto registrato e l'utente sta visualizzando le informazioni di tracciamento del lotto di produzione prodotto dalla compagnia.
Sequenza	• Il caso d'uso inizia quando l'utente seleziona "Notarizza" di uno step visualizzato nel tracciamento di un lotto di produzione (CU34) • Il sistema salva su un registro immutabile una descrizione dello step • Il sistema attende l'operazione di salvataggio • Il sistema allega allo step che si sta notarizzando un riferimento univoco all'operazione di salvataggio nel registro immutabile
Post-condizioni	L'identificativo univoco è visualizzato in CU34

Tabella 5: Caso d'uso 3a - Notarizzazione di uno step di processo

Tracciamento di un lotto (utente non registrato))	
ID	CU4
Attori	Utente non registrato
Precondizioni	TODO
Sequenza	TODO
Post-condizioni	TODO

Tabella 6: Caso d'uso 4 - Tracciamento di un lotto (utente registrato

Trasferimento di un lotto (atomico) [INCLUDE trasferimento])	
ID	CU5
Attori	Utente registrato che opera per conto di una compagnia A
Precondizioni	• L'utente è autenticato in piattaforma • Esiste una compagnia B registrata in piattaforma • La compagnia A ha un lotto di prodotto registrato
Sequenza	• Il caso d'uso inizia quando l'utente seleziona un lotto e clicca su "Trasferisci" • L'utente seleziona la compagnia destinataria, la quantità di prodotto da inviare e sceglie il tipo di trasferimento atomico • Il sistema controlla che i dati inseriti siano corretti • Se la quantità di prodotto è sufficiente allora C6x • Se la quantità di prodotto non è sufficiente • Il sistema avvisa l'utente • Il caso d'uso termina
Post-condizioni	• Se C6x – Il lotto creato è visibile alla compagnia destinataria – Il lotto di partenza ha la quantità aggiornata • Altrimenti – Il lotto di partenza non ha subito variazioni – La compagnia destinataria non visualizza nuovi lotti
Flusso alternativo	Il caso d'uso può ripetersi finché l'utente non seleziona una quantità di risorse che ha effettivamente a disposizione

Tabella 7: Caso d'uso 5 - Trasferimento di un lotto (atomico) [INCLUDE trasferimento]

Trasferimento di un lotto (atomico) [INCLUDE trasferimento])	
ID	CU6x
Attori	–
Precondizioni	–
Sequenza	• Il caso d'uso inizia quando al sistema arriva una richiesta di trasferimento di un lotto o porzione di esso dal proprietario attuale verso un nuovo proprietario • Il sistema crea un nuovo lotto a partere dalle informazioni del lotto originale e lo assegna alla compagnia destinataria • Il sistema aggiorna la quantità di prodotto nel lotto originale • Il sistema rende immutabile il trasferimento (notarizzazione) • Il sistema allega la notarizzazione al trasferimento
Post-condizioni	–

Tabella 8: Caso d'uso 6x - Trasferimento

Trasferimento di un lotto (con accettazione) [INCLUDE trasferimento]

ID	CU6
Attori	• Utente A registrato che opera per conto di una compagnia A • Utente B registrato che opera per conto di una compagnia B
Precondizioni	• L'utente A è autenticato in piattaforma • L'utente B è autenticato in piattaforma • La compagnia A ha un lotto di prodotto registrato

Sequenza	• Il caso d'uso inizia quando l'utente seleziona un lotto e clicca su "Trasferisci" • L'utente seleziona la compagnia destinataria, la quantità di prodotto da inviare e sceglie il tipo di trasferimento con accettazione • Il sistema controlla che i dati inseriti siano corretti • Se la quantità di prodotto non è sufficiente – Il sistema avvisa l'utente • Se la quantità di prodotto è sufficiente – Il sistema blocca la risorsa da trasferire – Il sistema notifica all'utente B la presenza di un trasferimento in ingresso (casi d'uso CU7 e CU9) – Il sistema rende annullabile trasferimento da parte dell'utente A (caso d'uso CU8) – Se si verifica CU7 allora CU6x 8 – Se si verifica CU9 o CU8 ∗ Il sistema ripristina la quantità di prodotto bloccata nel lotto originale ∗ Il caso d'uso termina
Post-condizioni	• Se C6x 8 – Il lotto creato è visibile alla compagnia destinataria – Il lotto di partenza ha la quantità aggiornata • Altrimenti – Il lotto di partenza non ha subito variazioni – La compagnia destinataria non visualizza nuovi lotti

Tabella 9: Caso d'uso 6 - Trasferimento di un lotto con accettazione [INCLUDE C6x 8 Trasferimento]

Accettazione di un trasferimento)	
ID	CU7
Attori	Utente registrato che opera per conto della compagnia destinataria del trasferimento
Precondizioni	La compagnia per cui opera l'utente registrato ha un trasferimento in ingresso di tipo "con accettazione"
Sequenza	• Il caso d'uso inizia quando l'utente visualizza la lista di trasferimenti della compagnia per cui opera • L'utente seleziona accetta un trasferimento di tipo con accettazione e • il sistema notifica l'accettazione alla compagnia mittente
Post-condizioni	Inizia CU6x 8

Tabella 10: Caso d'uso 7 - Accettazione di un trasferimento

Ritiro di un trasferimento)	
ID	CU8
Attori	Utente registrato che opera per conto della compagnia mittente del trasferimento
Precondizioni	• La compagnia per cui opera l'utente registrato ha un trasferimento in ingresso di tipo "con accettazione" • Il trasferimento non è stato ancora accettato dalla compagnia destinataria
Sequenza	• Il caso d'uso inizia quando l'utente visualizza la lista di trasferimenti della compagnia per cui opera • L'utente seleziona Ritira un trasferimento di tipo con accettazione • Il sistema notifica il ritiro alla compagnia destinataria • Il sistema aggiorna la quantità del lotto originario
Post-condizioni	–

Tabella 11: Caso d'uso 8 - Ritiro di un trasferimento

Rifiuto di un trasferimento)	
ID	CU9
Attori	Utente registrato che opera per conto della compagnia destinataria del trasferimento
Precondizioni	La compagnia per cui opera l'utente registrato ha un trasferimento in ingresso di tipo "con accettazione"
Sequenza	• Il caso d'uso inizia quando l'utente visualizza la lista di trasferimenti della compagnia per cui opera • L'utente seleziona rifiuta un trasferimento di tipo con accettazione • Il sistema notifica il rifiuto alla compagnia mittente • Il sistema aggiorna la quantità del lotto originario
Post-condizioni	–

Tabella 12: Caso d'uso 9 - Rifiuto di un trasferimento

Notarizzazione di un documento)	
ID	CU10
Attori	Utente registrato in piattaforma
Precondizioni	Utente autenticato e in possesso di un documento in formato pdf o jpg/png da notarizzare
Sequenza	• Il caso d'uso inizia quando l'utente seleziona "Carica documento" • L'utente sceglie dal suo dispositivo un documento nei formati consentiti (pdf, jpg o png) • L'utente clicca su carica • Il sistema effettua il caricamento della risorsa • Il sistema salva un'impronta digitale della risorsa all'interno del registro immutabile • Il sistema memorizza un riferimento univoco al salvataggio del punto precedente • Il sistema restituisce un indirizzo web per la verifica della transazione sul registro immutabile
Post-condizioni	• La risorsa è disponibile per il download • L'operazione di notarizzazione è pubblicamente verificabile

Tabella 13: Caso d'uso 10 - Notarizzazione di un documento

Avvio di una spedizione)	
ID	CU11
Attori	Utente registrato in piattaforma
Precondizioni	Utente autenticato in piattaforma, esiste una transazione completata (accettata o atomica) avviata dalla compagnia per cui opera l'utente e non è stato ancora disposto il trasferimento per la transazione
Sequenza	• Il caso d'uso inizia quando l'utente visualizza la lista delle transazioni • Se l'utente seleziona una transazione completata e clicca su "Invia" – Il sistema cerca un camion disponibile – Se ci sono camion disponibili ∗ il sistema associa il camion al lotto oggetto della transazione ∗ il sistema aggiorna periodicamente le letture dei sensori associati al camion ∗ il sistema salva in modo immutabile le letture dei sensori – Se non ci sono camion disponibili ∗ Il sistema notifica un errore
Post-condizioni	Si può vedere la lista delle letture dei sensori associati al camion

Tabella 14: Caso d'uso 11 - Avvio di una spedizione

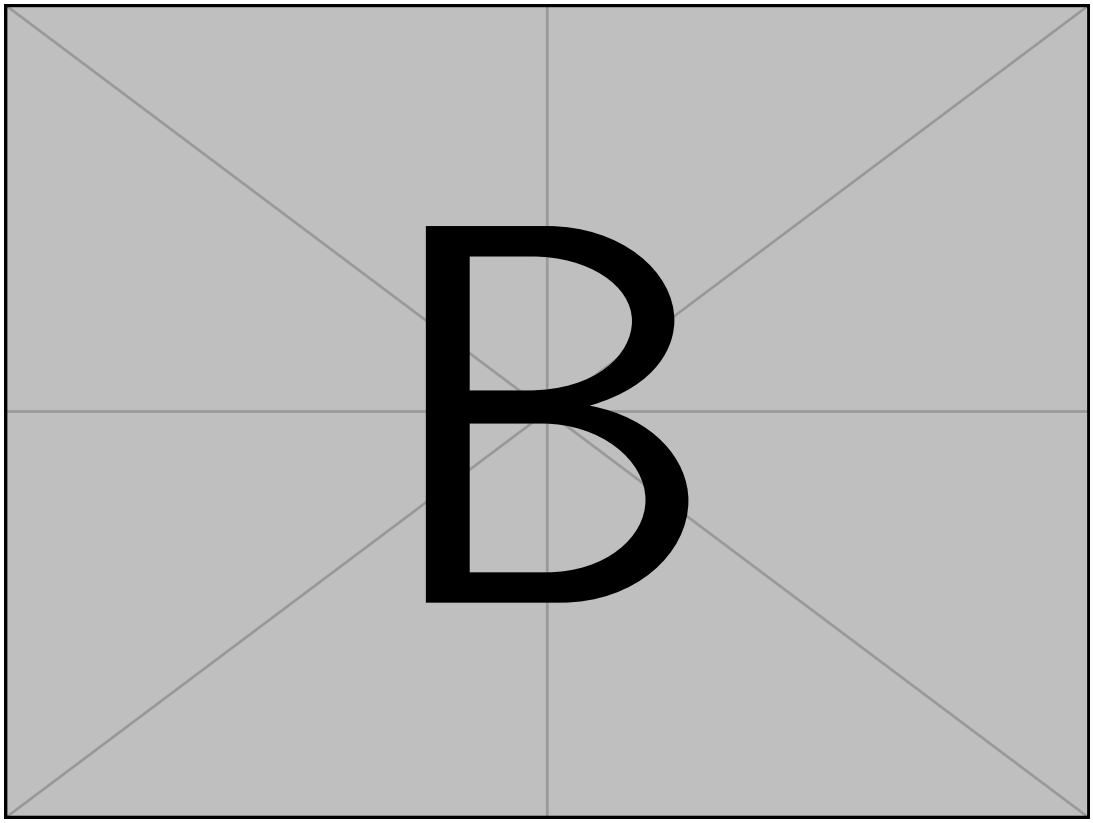

Figura 4: Diagramma dei casi d'uso

4.1.3 Persistenza

La piattaforma che si sta sviluppando utilizza un sistema di gestione della persistenza che è di tipo relazionale. Nello specifico utilizza il database managment system MariaDB con Hibernate come strato di comunicazione con l'applicazione. Hibernate implementa le API Java Persistence (JPA).

Hibernate è un ORM, un Object/Relational Mapping, un modulo software che gestisce la persistenza in contesti in cui si utilizza una base di dati relazionale.

Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities. It can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC. Hibernate's design goal is to relieve the developer from 95% of common data persistence-related programming tasks by eliminating the need for manual, hand-crafted data processing using SQL and JDBC. However, unlike many other persistence solutions,

Hibernate does not hide the power of SQL from you and guarantees that your investment in relational technology and knowledge is as valid as always. [8]

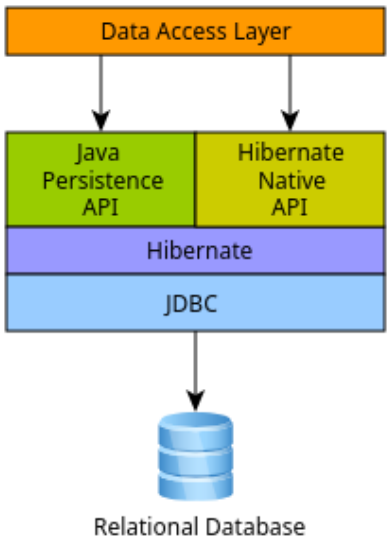

Figura 5: Dove si colloca Hibernate? [9]

4.1.4 Definizione del modello di dominio

La definizione di un modello di dominio a partire dai concetti principali che caratterizzano il modello di business individuato per l'applicazione. Sono stati individuati i seguenti concetti propri dell'ambito applicativo. Entità e attributi:

- Batch
  - Descrizione: Il lotto rappresenta un'unità di produzione di un particolare tipo di prodotto, in una determinata quantità.
  - Attributi: id, batchId, quantity, isFinal, productionDate, processType
- ProductType
  - Descrizione: Il tipo di prodotto è un generico prodotto inteso come l'astrazione, la generalizzazione, di lotti di produzione.
  - Attributi: id, name, unity, state
- Recipe (di ProductType)

  - Descrizione: Lista di ingredienti che compongono in generico prodotto. E' il modello da utilizzare quando si materializza la ricetta vera e propria durante la produzione di un lotto.
  - Attributi: id, note, recipeRow

- RecipeBatch (di Batch)
  - Descrizione: Lista di ingredienti effettivamente utilizzati nella produzione di un lotto di produzione.
  - Attributi: id, note, recipeRow

- ProductionProcess (di ProductType)
  - Descrizione: Lista di passi da seguire durante il processo produttivo del tipo cui l'entità si riferisce
  - Attributi: id, note, steps

- ProductionProcessBatch (di Batch)
  - Descrizione: Lista di passi da seguire durante il processo produttivo del tipo cui l'entità si riferisce
  - Attributi: id, note, steps

- Document
  - Descrizione: Risorsa che rappresenta un certificato, un'analisi o una generica attestazione da legare al lotto di produzione
  - Attributi: id, title, description, link, path

- Notarize
  - Descrizione: Entità che modella il concetto di notarizzazione di una qualunque risorsa all'interno della piattaforma. Consente di raccogliere insieme una o più transazioni reali sulla blockchain che hanno un legame logico tra di loro.
  - Attributi: id, hash, notarizedAt, data

- Transfer
  - Descrizione: Rappresentazione del trasferimento di un Batch o porzione di esso tra due Company registrate in piattaforma, si in modo atomico che con accettazione.

  - Attributi: id, type, status, oldBatchID, newBatchID, quantity, unity, companySenderID, companySenderUsername, companyRecipientID, companyRecipientUsername, transferDateStart, lastUpdate

- Transport
  - Descrizione:
  - Attributi: id, truckId, batchId, dateStart, dateEnd, location, companyFrom, companyTo

- Truck:
  - Descrizione: Modellazione del mezzo fisico (camion) che effettua il trasporto, ipotetico luogo dove si collocano i sensori che si vuole utilizzare
  - Attributi: id, lastSensorsUpdate

- UserInfo
  - Descrizione: Entità che rappresenta l'utente in piattaforma. Ha un riferimento al'auth server di Oauth2s.
  - Attributi: id, username, keycloakUsername, keycloakId, email, firstName, lastName

- Company
  - Descrizione: Modellazione dell'azienda che produce lotto di prodotti e che effettua operazioni su di essi in piattaforma.
  - Attributi: id, name

- ChainTransaction
  - Descrizione: Rappresenta la transazione sulla blockchain
  - Attributi: id, txId

- Location
  - Descrizione:
  - Attributi:

Relazioni:

- Company - Batch:
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Ogni Company può avere da 0 a n Batch prodotti. Ogni Batch ha una ed una sola Company che lo produce.
  - Ruolo: Produttore
- Company - Batch:
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Ogni Company può avere da 0 a n Batch detenuti. Ogni Batch ha una ed una sola Company che lo detiene.
  - Ruolo: Proprietario
- ProductType - Batch:
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Ogni ProductType può avere uno o più Batch che lo materializza. Un Batch ha esattamente un ProductType.
- Location - Batch;
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Ogni Location può essere il luogo di produzione di zero o più Batch. Un Batch ha esattamente una Location.
- Document - Batch;
  - Cardinalità: molti a molti con partecipazione opzionale
  - Descrizione: Ogni Document puà essere legato a più Batch, ogni Batch può avere collegati zero o più Document.
- RecipeBatch - Batch:
  - Cardinalità:
  - Descrizione:
- ProductionProcessBatch - Batch:
  - Cardinalità:

  - Descrizione:

- ProductType - Company:
  - Cardinalità: Molti a molti
  - Descrizione: Ogni tipo di prodotto può essere legato ad una o più Company ed ogni Company può utilizzare uno o più tipi di prodotto

- UserInfo - Company:
  - Cardinalità: uno a molti con partecipazione obbligatoria
  - Descrizione: Un utente (UserInfo) ha esattamente una compagnia (Company) collegata. Ogni compagnia (Company) ha almeno un utente collegato.

- Notarize - Document:
  - Cardinalità: uno ad uno con partecipazione opzionale
  - Descrizione: Ogni Document ha alpiù una notarizzazione ed ogni Notarize può avere alpiù un Document

- Company - Document:
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Ogni Document da riferimento ad esattamente una Company. Una Company puoò avere zero o più Document collegati.

- Company - Notarize:
  - Cardinalità: uno a molti con partecipazione obbligatoria
  - Descrizione: Una Notarize ha esattamente una Company che l'ha creata ed una Company può creare zero o più Notarize.

- ChainTransaction - Notarize:
  - Cardinalità: uno a molti con partecipazione obbligatoria
  - Descrizione: Una Notarize ha almeno una ChainTransaction (transazione effettiva sulla blockchain). Ogni ChainTransaction fa riferimento ad una sola Notarize.

- ProductionStep - ProductionProcess
  - Cardinalità: uno a molti con partecipazione obbligatoria

  - Descrizione: Un processo produttivo di tipo (ProductionProcess) ha almeno uno step di produzione di tipo (ProductionStep). Ogni step (ProductionStep) fa parte di un singolo processo produttivo (ProductionProcess).

- ProductionStepBatch - ProductionProcessBatch
  - Cardinalità: uno a molti con partecipazione obbligatoria
  - Descrizione: Un processo produttivo di lotto (ProductionProcessBatch) ha almeno uno step di produzione di lotto (ProductionStepBatch). Ogni step (ProductionStepBatch) fa parte di un singolo processo produttivo (ProductionProcessBatch).

- ProductionStepBatch - Notarize
  - Cardinalità: uno ad uno con partecipazione opzionale
  - Descrizione: Ogni processo produttivo di lotto (ProductionStepBatch) può avere al più un atto di notarizzazione (Notarize).

- Recipe - ProductType:
  - Cardinalità:
  - Descrizione:

- ProductionProcess - ProductType:
  - Cardinalità:
  - Descrizione:

- Recipe - RecipeRow:
  - Cardinalità: uno a molti con partecipazione obligatoria
  - Descrizione: Ogni ricetta di tipo (Recipe) ha almeno un ingrediente di tipo(RecipeRow).

- RecipeBatch - RecipeRowBatch:
  - Cardinalità: uno a molti con partecipazione obligatoria
  - Descrizione: Ogni ricetta di lotto (RecipeBatch) ha almeno un ingrediente di lotto (RecipeRowBatch).

- RecipeRow - ProductType:

  - Cardinalità: uno a molti con partecipazione obbligatoria
  - Descrizione: Ogni riga di ricetta di tipo (RecipeRow) si riferisce ad esattamente un tipo di prodotto (ProductType). Ogni ProductType può fare parte di zero o più ricette di tipo.

- RecipeRowBatch - Batch
  - Cardinalità: uno a molti con partecipazione obbligatoria
  - Descrizione: Ogni riga di ricetta di lotto (RecipeRowBatch) si riferisce ad esattamente un lotto di prodotto (Batch). Ogni Batch può fare parte di zero o più ricette di lotto.

- SensorsLog - Notarize
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Gruppi di item di log dei sensori (SensorsLog) possono fare riferimento ad uno stesso item di notarizzazione (Notarize) se sono stati raggruppati e notarizzati nello stesso momento, per esempio per letture ravvicinate nel tempo.

- ChainTransaction - Transfer
  - Cardinalità: uno a molti con partecipazione opzionale
  - Descrizione: Ogni trasferimento di lotti (Transfer) si riferisce ad una o più transazioni effettive sulla chain (ChainTransaction). Per esempio, il tipo con accettazione prevede tre transazioni.

- Truck - Company
  - Cardinalità: uno a molti
  - Descrizione: Ogni Truck fa riferimento ad una sola Company

- Truck - MyDt
  - Cardinalità: uno a molti
  - Descrizione: Ogni Truck ha una lista di gemelli digitali collegati

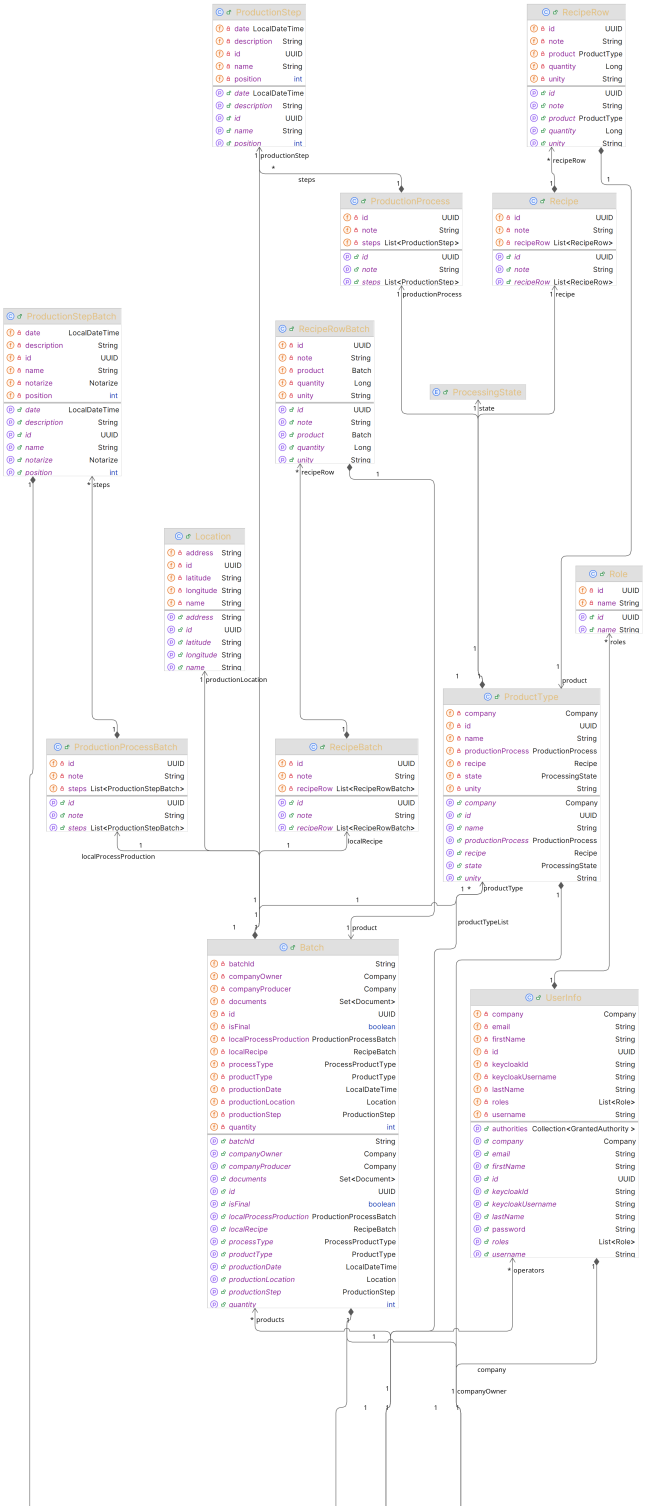

Figura 6: Diagramma UML classi - parte 1 di 2

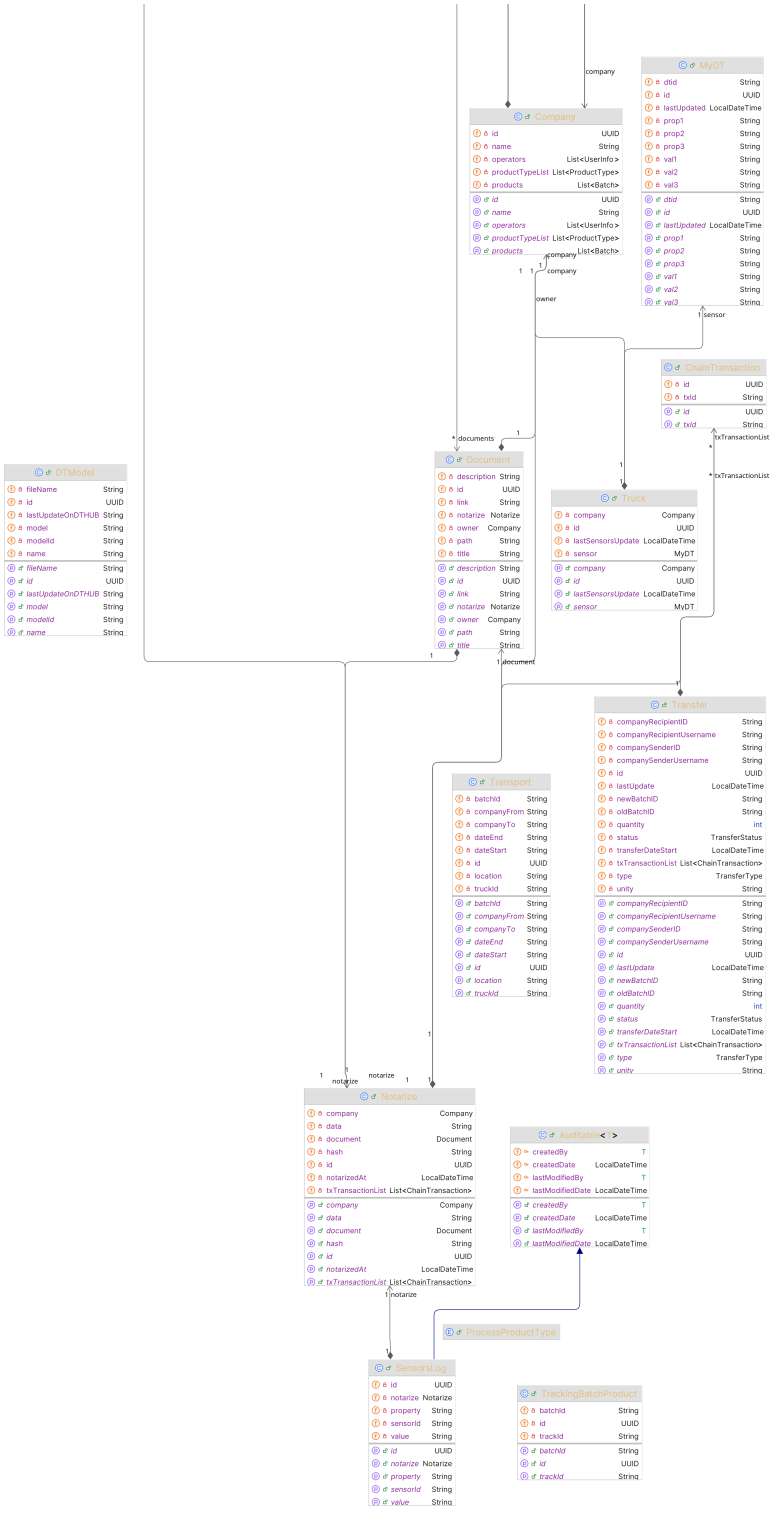

Figura 7: Diagramma UML classi - parte 2 di 2

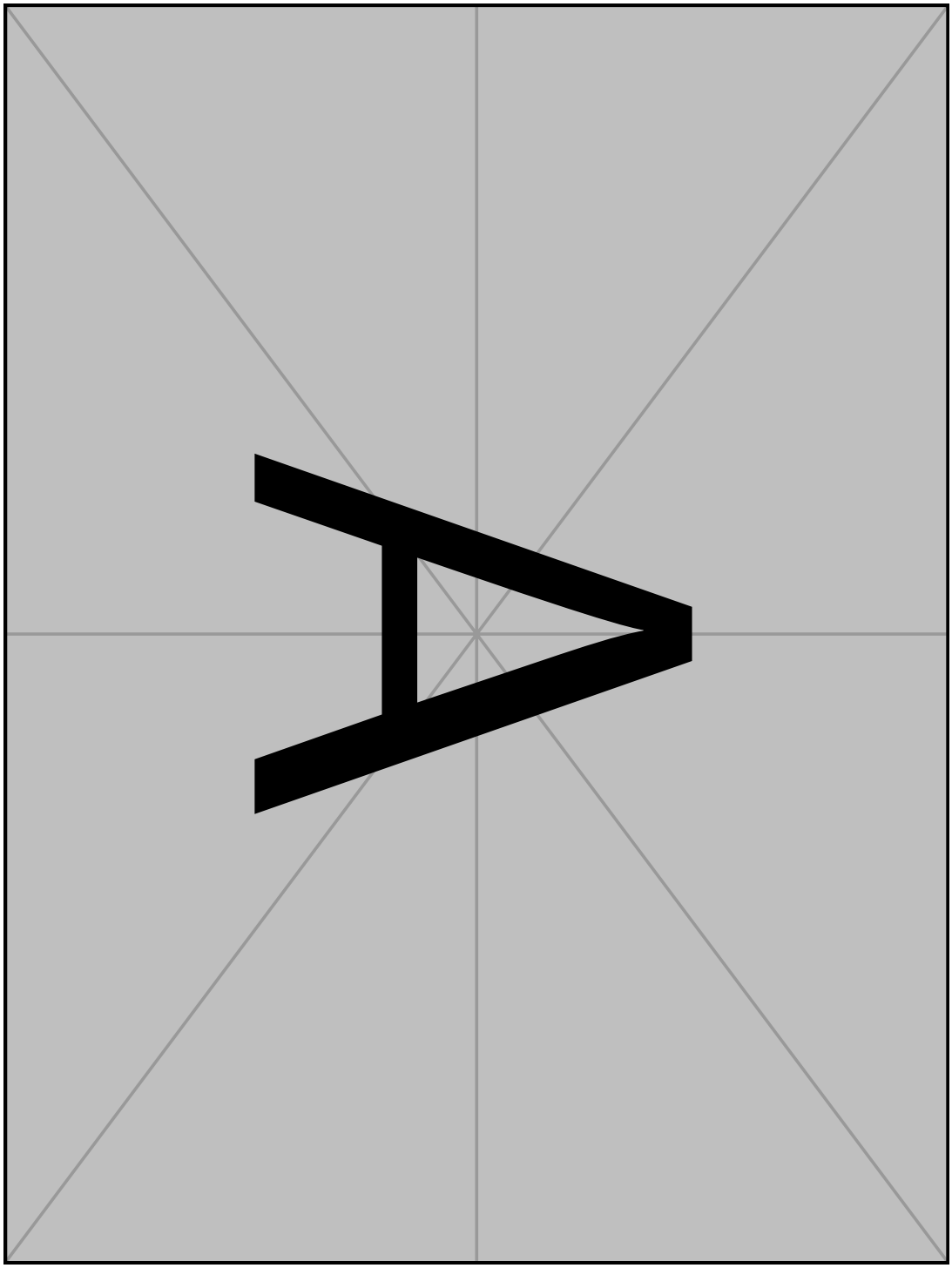

Figura 8: Schema Entità-Relazione

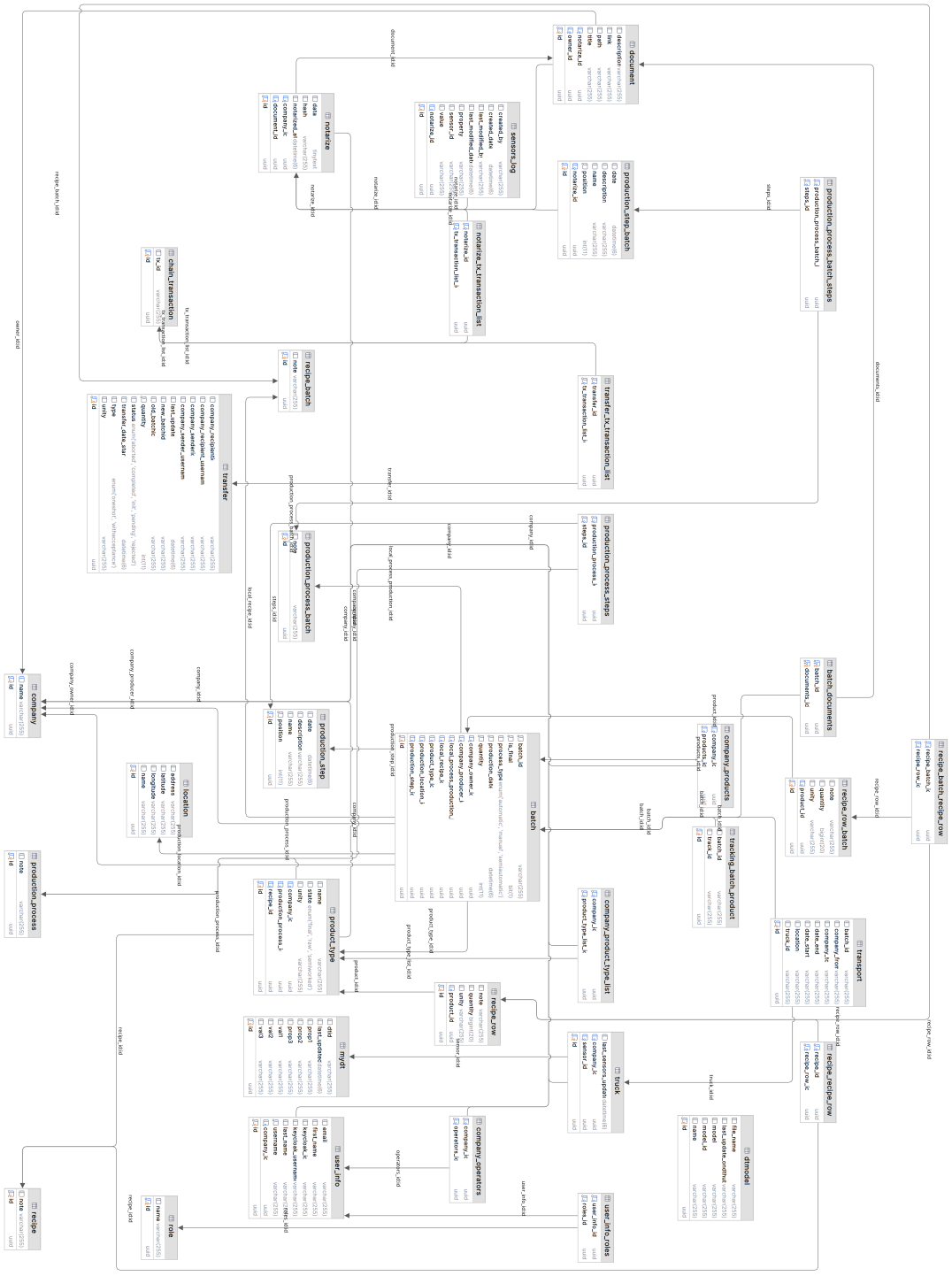

Figura 9: Schema logico base di dati

4.1.5 Answar Set Programming

Il modulo di answer set programming utilizza, tramite un wrapper non ufficiale rilasiato con licenza open source, il pacchetto clingo di Potasco [1] che rappresenta una soluzione completa all'esecuzione di programmi ASP perché combina insieme entrambe le componenti necessarie ad una esecuzione corretta di un programma ASP, ovvero il grounder gringo che il solutore clasp.

Ad un livello precedente troviamo un servizio middleware costruito ad hoc che prepara le strutture dati Java in programmi eseguibili da clingo, quindi compie un operazione di serializzazione.

L'output di clingo è poi opportunamente gestito da un altra porzione del servizio middleware che si occupa della deserializzazione della stringa che rappresenta l'output.

4.1.6 Gemello digitale

I gemelli digitali utilizzati nel progetto sono stati implementati sfruttando la piattaforma Digital Twin di Microsoft Azure. Questo servizio è una piattaforma a servizi (PaaS) che consente la gestione di gemelli digitali sia autonomi che eventualmente integrati con altri servizi, non necessariamente limitati al contesto Azure, poiché vengono esposti degli endpoint che, al netto di un'eventuale stretta sulle politiche di accesso definita dall'utente, sono pubblici ed implementano lo standard RESTAPI. L'integrazione e l'utilizzo dei gemelli digitali in una applicazione Java necessita di una componente software specializzata. In questo caso ci si è diretti verso il Software Development Kit (SDK) fornito da Microsoft Azure. L'SDK fornisce un approccio modulare consentendo l'adozione solo delle parti necessarie ed in particolare si è utilizzato il modulo "Azure IoT Digital Twins client library for Java".

4.1.7 Smart contract e blockchain

Gli smart contract posso essere visti come le classi di un programma orientato agli oggetti, Java ad esempio. Si possono scrivere in diversi linguaggi di programmazione, a seconda della blockchain utilizzata. In questo progetto di tesi si utilizza Polygon e gli smart contract su Polygon si scrivono in Solidity.

> Solidity è un linguaggio di alto livello orientato agli oggetti per l'implementazione di contratti intelligenti. I contratti intelligenti sono programmi che regolano il comportamento dei conti all'interno dello stato di Ethereum. [...] è un linguaggio a parentesi graffe progettato per la macchina virtuale di Ethereum (EVM). È

[1] https://github.com/potassco/clingo

influenzato da C++, Python e JavaScript. [...] Solidity è tipizzato staticamente, supporta l'ereditarietà, le librerie e i tipi complessi definiti dall'utente [2] [15]

Durante la fase preliminare di scouting delle librerie sono stati individuati alcuni tool di interesse. Si e' individuato il seguente stack tecnologico per la compilazione e la distribuzione di contratti scritti in Solidity ed eseguibili all'interno di una Ethereum Virtual Machine.

- **Solidity** scrittura del contratto
- **HardHat** compilazione, distribuzione e test dei contratti
- **Ether.js** client Ethereum per NodeJS

Inoltre affinché la piattaforma supporti pienamente le operazioni di lettura e scrittura su un registro distribuito è necessario utilizzare alcuni componenti software aggiuntivi. Nello stack tecnlogico adottato è possibile inserire a questo proposito il progetto Web3j, una libreria java che supporta gli SmartContract e consente di interagire con la rete Etherium tramite un client JSON-RPC, utilizzando chiamate API su protocollo HTTP o IPC. [11]

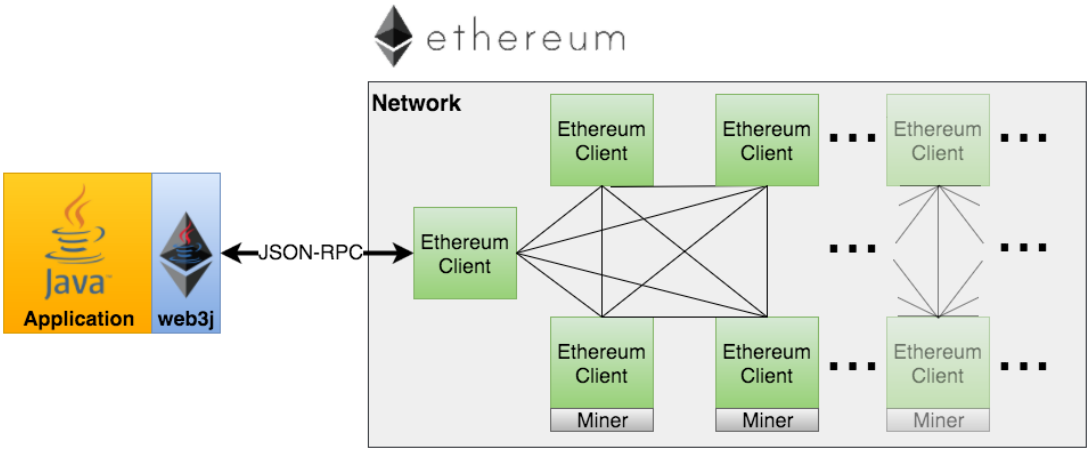

Figura 10: Web3j [11]

Essendo Java un linguaggio fortemente tipizzato necessita di un tipo, di una classe, che rappresenti lo smart contract per poter essere utilizzato all'interno dell'applicazione. La libreria offre un tool a riga di comando che consente la generazione

[2] Tradotto con DeepL.com (versione gratuita)

automatica di un wrapper, una classe java, per ogni contratto scritto in solidity che si vuole utilizzare.

Benché Web3j supporti ogni aspetto del ciclo di vita degli smart contract, si è deciso di adottare una soluzione software differenze per la distribuzione, il testing e la verifica dei contratti, creando di fatto un modulo separato e isolato dal resto dell'applicazione e lasciando alla parte Spring-Web3j solo la responsabilità dell'utilizzo vero e proprio dei contratti.

Il modulo di creazione dei contratti risiede in ambiente NodeJS ed utilizza il framework HardHat.js.

Hardhat è un framework modulare composto da diverse parti tra loro interconnesse ma comunque autonome. Nella documentazione del progetto si definisce il framework come un ambiente di sviluppo. Hardhat is a development environment for Ethereum software. It consists of different components for editing, compiling, debugging and deploying your smart contracts and dApps, all of which work together to create a complete development environment. [5]

Le parti principali che compongono l'ambiente di sviluppo sono:

- runner: è il componente principale di hardhat, consente di compilare, distribuire, testare e verificare gli smart contract
- ignition: consente di utilizzare un approccio dichiarativo alla gestione degli smart contract
- net: una rete locale di test
- estensione per visual studio code

Degna di nota è sicuramente la parte di testing degli smart contract che attraverso l'integrazione di librerie ben note in ambito Test Drive Development (TDD)su NodeJs quali Mocha [3] e Chai [4] consente un rapido sviluppo di smart contract sicuri e testati. I contratti utilizzati in piattaforma sono stati sviluppati seguendo un approccio TDD, raggiungendo un valore di coverage superiore al 90

Una delle motivazioni principali che hanno spinto verso questa scelta di separazione è stata il pieno supporto di questo altro stack tecnologico alla libreria OpenZeppelin, la cui adozione riveste un'importante scelta in ambito di sicurezza degli smart contract. Basti pensare che uno smart contract che non adotta

[3] https://mochajs.org/
[4] https://www.chaijs.com/

strategie difensive di alcun tipo può essere utilizzato da chiunque ne conosca l'indirizzo (pubblico!). Questo sicuramente non è un comportamento voluto o desiderato dalla maggior parte dei contesti applicativi. Un primo rimedio che offre OpenZeppelin è la super classe Owanable che se implementata consente di porre vincoli sull'utente che può effettivamente utilizzare il contratto, anche se ne conosce l'indirizzo. Esistono livelli più avanzati di gestione degli accessi e dei ruoli ed OpenZeppelin offre supporto anche a questi casi d'uso.

4.1.8 Riflessioni sulla sicurezza degli smart contract

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.24;
3
4  // Import Ownable from the OpenZeppelin Contracts library
5  import "@openzeppelin/contracts/access/Ownable.sol";
6  import
   ↪ "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
7
8  contract Hash is Initializable {
9      struct SignData {
10         bytes32 hashSigned;
11         uint256 timestamp;
12         string data;
13     }
14     mapping(bytes32 => SignData) public signData;
15     mapping(bytes32 => bool) public keyExists;
16
17     event HashSigned(bytes32 indexed hash, address indexed
       ↪ signer, SignData data);
18     function initialize() public {}
19
20     function signHash(bytes32 hash, string memory data) public {
21         require(hash != bytes32(0), "Hash cannot be zero");
22         require(!keyExists[hash], "Hash already signed");
23
24         keyExists[hash] = true;
```

```
25          signData[hash] = SignData({
26              hashSigned: hash,
27              timestamp: block.timestamp,
28              data: data
29          });
30
31          emit HashSigned(hash, msg.sender, signData[hash]);
32      }
33
34      function isHashSigned(bytes32 hash) public view returns
        ↪  (bool) {
35          return keyExists[hash];
36      }
37
38      function getSignData(
39          bytes32 dataHash
40      ) public view returns (bytes32, uint256, string memory) {
41          SignData memory record = signData[dataHash];
42          require(keyExists[dataHash], "Hash not signed");
43          return (record.hashSigned, record.timestamp,
            ↪  record.data);
44      }
45  }
```

Figura 11: Contratto di notarizzazione - versione finale

4.2 Implementazione - server side

Sono state effettuate delle scelte a livello implementativo. - Ultima versione Java attualmente disponibile (22) in versione OpenJDK. - Utilizzo di Keycloak in versione dockerizzata

4.2.1 Autenticazione

L'autenticazione degli utenti è gestita dal server autoritativo Keycloak. In un contesto OAuth2 il componente Spring riveste il ruolo di resource server, Key-

cloak è l'authentication server, le risorse sono le informazioni veicolate tramite le chiamate alle API[5] ed in fine la componente d'interfaccia, che viene eseguita nel browser dell'utente, è il client.

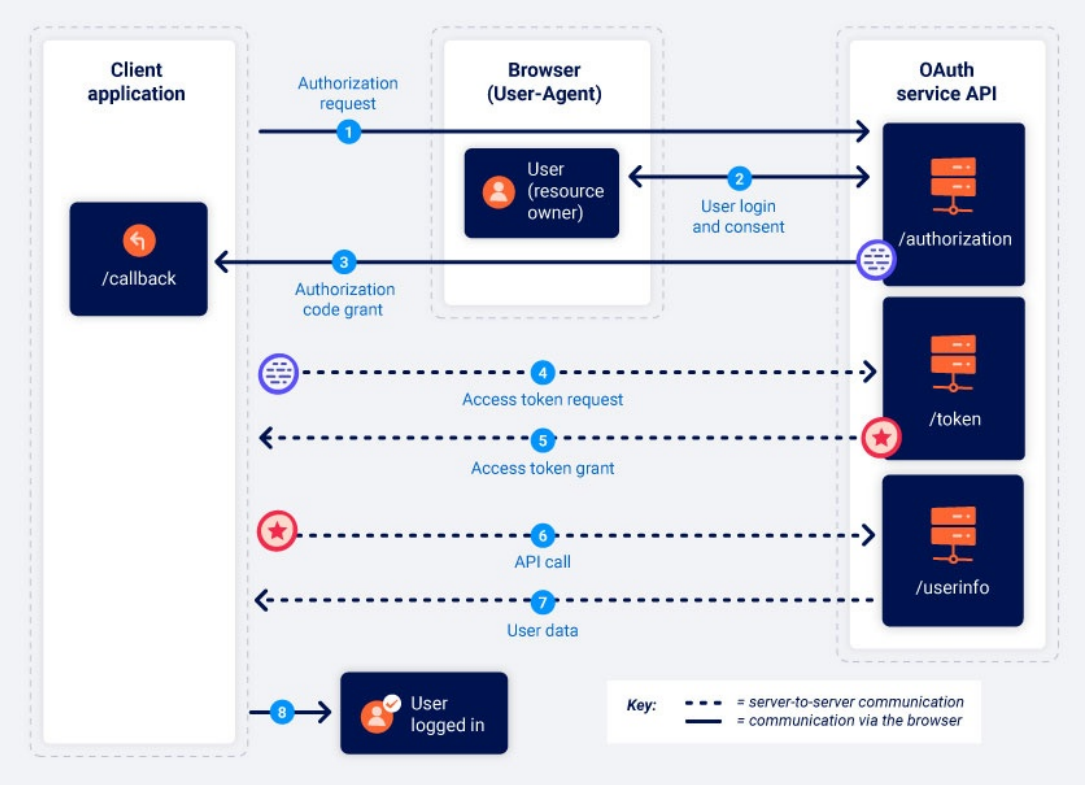

Figura 12: Authorization code grant type - da [13]

Keycloak è implementato utilizzando la tecnologia di conteinerizzazione Docker. I docker sono container definiti leggeri poiché l'isolamento che offrono riguarda i processi. In una macchina virtuale, definita conteinerizzazione pesante, il grado di isolamento è maggiore, visto che si instanzia un sistema operativo da zero e la condivisione tra sistema operativo ospitante (host) e sistema operativo emulato (guest) è limitata alle risorse hardware. Con le macchine virtuali si massimizza l'isolamento ma si massimizza anche l'overhead inteso come dispendio di energie computazionali spese nella gestione di elementi *duplicati*, altrimenti già disponibili nel sistema host. [21]

[5] Application Programming Interface - per una lista degli endpoints disponibili vedere ??

Listing 1: Keycloak - Comando di avvio

In questo contesto prototipale, il server autoritativo è stato avviato in modalità di sviluppo. Le limitazioni e le semplificazioni apportate riguardano la gestione del TLS e della mutua autenticazione. In fase di sviluppo non è richiesta la configurazione di un keystore java.

Il comando di avvio utilizzato è una personalizzazione del comando che si può trovare nella documentazione ufficiale [18]. L'adattamento effettuato serve a caricare all'avvio il file di configurazione di progetto. Nel file di configurazione sono esplicitati tutte le impostazioni di *realm*. Il realm è l'ambito di autenticazione, lo scope per dirla in termini strettamente di codice, per cui quando un utente si registra o viene creato esso è registrato o creato all'interno di un realm. In questo caso la piattaforma ha un suo realm. Il resource server e il client, rispettivamente componente Spring e componente Angular, sono registrati sul server autoritativo come client del ream *iotchain*.

4.2.2 Application Programming Interface (API)

Sono state definite i senguenti endpoints:

- `GET /api/v1/company/batch`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/company/batch`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/batch/id`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:

- `GET /api/v1/company/batch/id/track`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/product-type`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/company/product-type`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/client`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transfer`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/company/transfer`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transfer/{{batch_id}}`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:

- `GET /api/v1/company/transfer/{{trans_id}}/abort`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transfer/{{trans_id}}/accept`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transfer/{{trans_id}}/reject`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/doc`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/doc/notarize/{{doc_id}}`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/company/doc/upload`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/doc/{{doc_id}}`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:

- `GET /api/v1/company/doc/{{doc_id}}/resource`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/public/doc/check/hash`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transport`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/company/transport`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transport/{{batch_id}}`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/transport/{{transport_id}}/truck`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/truck`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:

- `GET /api/v1/company/truck/update`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `GET /api/v1/company/notarization`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:
- `POST /api/v1/company/notarization/step/{{step_id}}`
  Descrizione:
  Tipo di accesso:
  Input:
  Output:

4.2.3 Persistenza

L'implementazione della persistenza da un punto di vista fisico e di basso livello è stata demandata all'ORM (Object Relational Mapping) utilizzato, Hibernate, nella versione XXX. Il DBMS usato in questa fase prototipale è MariaDB 11.4.2 installato in modo nativo sulla macchina di sviluppo (quindi non conteinerizzato).

Per ogni concetto evidenziato in 4.1.4 è stata creata una classe Java marcata come entità usando l'annotazione `@Entity` del layer di persistenza di Jakarta (specifica implementata da Hibernate).

Ogni entità del modello di dominio estende la classe astratta `Auditable` così da permettere un tracciamento dettagliato delle modifiche alle entità stesse. Ogni entità utilizza come chiave primaria un identificativo univoco di tipo alfanumerico, uno Universally unique identifier (UUID). Gli UUID posso essere di diversi tipi, il tipo utilizzato è il tipo 4, il tipo randomico. Questo tipo di dato è utilizzabile in maniera nativa perché MariaDB dalla versione 10.7 [19] supporta nativamente il tipo UUID.

4.2.4 Programmazione Asincrona e thread virtuali

Il server implementato è un server che ha un funzionamento sincrono e bloccante. Ovvero il sistema riceve e gestisce la connessione in ingresso e su di questa si blocca, attendendo l'esito della computazione prima di chiudere la connessione stessa. In un contesto simile è complesso gestire la comunicazione con la blockchain o con altri servizi quali la piattaforma di gestione dei digital twin. Si è rivelato indispensabile attivare la possibilità di effettuare chiamate asincrone che funzionano in un ottica multithread: la connessione in ingresso è gestita e chiusa dall'application server (Tomcat) quasi immediatamente, senza dover attendere il termine della chiamata asincrona. La chiamata a funzione asincrona esegue la computazione e poi salva il risultato della computazione stessa sulla base di dati, rendendo persistente l'informazione. Eliminando di fatto la necessità per l'utente di attendere il termine della connessione che durante alcuni test di scrittura sulla chain, con rete sovraffollata, può superare i 10 minuti di attesa. Il risultato della scrittura sulla blockchain quando disponibile si può ottenere effettuando una semplice interrogazione sulla base di dati. Per una migliore gestione delle risorse sono stati utilizzati i thread virtuali, che la JVM mette a disposizione degli sviluppatori a partire dalla versione 21. I thread virtuali, a differenza dei thread di sistema, sono gestiti dalla JVM e non richiedono chiamate di sistema. Offro il vantaggio di poter essere sospesi dalla JVM e liberare il thread di sistema sottostante, quando per esempio si compiono operazioni I/O bloccanti, ed essere ripresi quando è possibile. Il thread di sistema liberato può eseguire altri thread virtuali. [2]

La configurazione utilizzata in piattaforma è la seguente:

```
// Visto che si sta usando Java > 21
// l'attivazione dei thread virtuali è sicuramente più semplice

// CONTROLLARE da quale versione di Java è vera questa cosa,
// sicuramente con la 22 funziona ma non sono sicura se inizia
// a funzionare dalla 20 o dalla 21!

// Basta inserire la seguente righe nelle applicazion.properties

// spring.threads.virtual.enabled=true
```

```java
// spring.thread-executor=virtual

// Configurazione
@EnableAsync
@Configuration
@ConditionalOnProperty(
  value = "spring.thread-executor",
  havingValue = "virtual"
)
public class ThreadConfig {
  @Bean
  public AsyncTaskExecutor applicationTaskExecutor() {
    return new
     ↪ TaskExecutorAdapter(Executors.newVirtualThreadPerTaskExecutor());
  }

  @Bean
  public TomcatProtocolHandlerCustomizer<?>
   ↪ protocolHandlerVirtualThreadExecutorCustomizer() {
    return protocolHandler -> {

      ↪ protocolHandler.setExecutor(Executors.newVirtualThreadPerTaskExecutor()
    };
  }
}
```

Figura 13: Configurazione Asincrona da [22]

Una funzione annotata come `@Bean` consente di utilizzate la funzione stessa in ogni porzione di codice, utilizzando la dependency injection e l'inversione del controllo forniti da Spring. Questi due Bean di configurazione specificano quale tipologia di Thread si vuole utilizzare, con il Tomcat fornito da Spring Boot e con le funzioni annotate come asincrone. I thread virtuali offrono prestazioni nettamente migliori in termini di richieste gestite nell'unità di tempo [22].

4.2.5 Answer Set Programming

Per poter utilizzare all'interno di un contesto più ampio un programma in AnswareSetProgramming (ASP) che sia dinamico, ovvero che utilizzi dati che cambiano nel tempo e provengono da computazioni effettuate in altri moduli del software, è necessario in un primo momento effettuare una mappatura della struttura dati in input al programma stesso e successivamente a comp

```
batch(batch_id_3, type2, 6).
batch(batch_id_4, type2, 1).
batch(batch_id_5, type2, 5).

batch(batch_id_10, type1, 3).
batch(batch_id_11, type1, 3).
batch(batch_id_12, type1, 3).

amount(2).
type(type2).
```

Figura 14: Programma ASP - Ricerca lotto - Esempio fatti in input

```
{ scelto(E, Q) } :- batch(E, X, Q), type(X).
tot(T) :- T = #sum { Q, E : scelto(E, Q) }.
:- amount(A), tot(B), A > B.
#minimize { 1, Q : scelto(E,Q) }.
#show scelto/2.
```

Figura 15: Programma ASP - Ricerca lotto

```
@Override
public String listToAtomArita3(String name, List<Triplet> value)
 ↪  {
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < value.size(); i++) {
   ↪  sb.append(name).append("(").append(value.get(i).getValue0()).append(",").ap
   ↪  ");
    }
```

```
  return sb.toString();
}
```

Figura 16: ASP - Mapping in ingresso

```
@Override
public List<Triplet> atomArita3ToList(String output, String atom)
 ↪  {
  List<Triplet> result = new ArrayList<>();
  Arrays.stream(output.split(" "))
    .filter(a -> a.startsWith(atom))
    .forEach(a -> {
      var x = a.replace(atom, "").replace("(", "").replace(")",
       ↪  "");
      String[] split = x.split(",");
      result.add(Triplet.with(split[0], split[1], split[2]));
    });
  return result;
}
```

Figura 17: ASP - Mapping in uscita

```
@Override
public List<String> solveFileAndString(String filename, String
 ↪  customString) throws IOException, URISyntaxException {
  List<String> results = new ArrayList<>();
  control = new Control("0");
  String prog = readFileAdString("asp/" + filename);
  control.add(prog);
  control.add(customString);
  control.ground();
  try (SolveHandle handle = control.solve(SolveMode.YIELD)) {
    while (handle.hasNext()) {
      Model model = handle.next();
      results.add(model.toString());
    }
  }
```

```
  control.close();
  return results;
}
```

Figura 18: ASP - Solutore

4.2.6 Gemello digitale

```
{
  "@id": "dtmi:iotchain:DigitalTwins:Truck;1",
  "@type": "Interface",
  "displayName": "Truck interface model",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Property",
      "name": "Location",
      "schema": "string"
    },
    {
      "@type": "Property",
      "name": "Temperature",
      "schema": "double"
    }
  ]
}
```

Figura 19: Modello di Gemello Digitale implementato

```
{
  "@id": "dtmi:iotchain:DigitalTwins:Truck;1",
  "@type": "Interface",
  "displayName": "Truck interface model",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
```

```
      "@type": "Property",
      "name": "Location",
      "schema": "string"
    },
    {
      "@type": "Property",
      "name": "Temperature",
      "schema": "double"
    }
  ]
}
```

Figura 20: Modello di Gemello Digitale implementato

```
@Override
public String createOneSensor(String sensorName, String dtName) {
  DTModel model =
   ↪ dTModelRepository.findByName(sensorName).orElseThrow();
  BasicDigitalTwin basicTwin = new BasicDigitalTwin(dtName)
    .setMetadata(new BasicDigitalTwinMetadata()
      .setModelId(model.getModelId()))
    .addToContents("Temperature", 0)
    .addToContents("Location", "none");
  BasicDigitalTwin basicTwinResponse =
   ↪ digitalTwinsClientSync.createOrReplaceDigitalTwin(dtName,
   ↪ basicTwin, BasicDigitalTwin.class);
  log.debug("Ho creato il sensore {} ",
   ↪ basicTwinResponse.getId());
  return basicTwinResponse.getId() != null &&
   ↪ !basicTwinResponse.getId().isEmpty() ?
   ↪ basicTwinResponse.getId() : dtName;
}
```

Figura 21: Creazione del gemello digitale

```
import com.azure.core.credential.TokenCredential;
import com.azure.digitaltwins.core.DigitalTwinsClient;
```

```java
import com.azure.digitaltwins.core.DigitalTwinsClientBuilder;
import com.azure.identity.DefaultAzureCredentialBuilder;

@Service
public class DtServiceImpl implements DtService {
  private final DigitalTwinsClient digitalTwinsClientSync;

  public DtServiceImpl() {
    this.digitalTwinsClientSync = new DigitalTwinsClientBuilder()
      .credential(new DefaultAzureCredentialBuilder().build())
      .endpoint(dturl)
      .buildClient();
  }
}
```

Figura 22: Creazione del gemello digitale

```java
// PollingConfig.java

@Component
@Slf4j
@RequiredArgsConstructor
public class PollingConfig implements Serializable {
  private final TransportService transportService;

  @Async
  @Scheduled(fixedDelayString = "${app.dt.interval-update-ms}",
   ↪  initialDelayString = "${app.dt.initial-update-ms}")
  public void poll() throws InterruptedException {
    log.info("Polling from DT Hub...");
    transportService.updateAllTransportDataFromDTHUb();
  }
}
```

```java

// TransportServiceImpl.java

@Override
@Async
public void updateAllTransportDataFromDTHUb() throws
 ↪  InterruptedException {
  List<Transport> transports = transportRepository.findAll();
  List<UUID> trucksId = new ArrayList<>();
  transports.stream().filter(el -> el.getTruckId() !=
   ↪  null).forEach(el ->
   ↪  trucksId.add(UUID.fromString(el.getTruckId())));
  List<Truck> trucks = truckRepository.findAllById(trucksId);
  List<MyDT> sensors = new ArrayList<>();
  trucks.forEach(el -> {
    el.setLastSensorsUpdate(LocalDateTime.now());
    sensors.add(el.getSensor());
  });
  dtService.updateSensors(sensors);
  sensors.forEach(el -> el.setLastUpdated(LocalDateTime.now()));
  myDTRepository.saveAll(sensors);
  truckRepository.saveAll(trucks);
}
```

Figura 23: Creazione del gemello digitale

```java
  @Override
  public Map<String, Object> getSensorData(String sensorId,
   ↪  List<String> props) {
    Map<String, Object> sensorData = new HashMap<>();

    var dt = digitalTwinsClientSync.getDigitalTwin(sensorId,
     ↪  Map.class);

    dt.forEach((key, value) -> {
      if (props.contains(key)) {
```

```
        sensorData.put((String) key, value);
      }
    });

    return sensorData;
  }

@Override
  @Async
  public void updateSensors(List<MyDT> sensors) {
    sensors.forEach(sensor -> {
      List<String> props = new ArrayList<>();

      if (sensor.getProp1() != null) {
        props.add(sensor.getProp1());
      }
      if (sensor.getProp2() != null) {
        props.add(sensor.getProp2());
      }
      if (sensor.getProp3() != null) {
        props.add(sensor.getProp3());
      }

      Map<String, Object> res =
       ↪  this.getSensorData(sensor.getDtid(), props);
      List<SensorsLog> sensorsLogs = new ArrayList<>();

      if (res.get(sensor.getProp1()) != null) {

         ↪  sensor.setVal1(String.valueOf(res.get(sensor.getProp1())));
        sensorsLogs.add(SensorsLog.builder()
          .sensorId(sensor.getDtid())
          .property(sensor.getProp1())
          .value(sensor.getVal1())
          .build());
      }
```

```
43
44      if (res.get(sensor.getProp2()) != null) {
45
            sensor.setVal2(String.valueOf(res.get(sensor.getProp2())));
46        sensorsLogs.add(SensorsLog.builder()
47          .sensorId(sensor.getDtid())
48          .property(sensor.getProp2())
49          .value(sensor.getVal2())
50          .build());
51      }
52
53      if (res.get(sensor.getProp3()) != null) {
54
            sensor.setVal3(String.valueOf(res.get(sensor.getProp3())));
55        sensorsLogs.add(SensorsLog.builder()
56          .sensorId(sensor.getDtid())
57          .property(sensor.getProp3())
58          .value(sensor.getVal3())
59          .build());
60      }
61      List<MyDT> sensorSaved = myDTRepository.saveAll(sensors);
62      List<SensorsLog> saved =
          sensorsLogRepository.saveAll(sensorsLogs);
63      try {
64        if (!saved.isEmpty()) {
65          notarizeService.notarize(saved);
66        }
67      } catch (NoSuchAlgorithmException e) {
68        throw new RuntimeException(e);
69      } catch (IOException e) {
70        throw new RuntimeException(e);
71      } catch (TransactionException e) {
72        throw new RuntimeException(e);
73      }
74    });
```

```
75 }
```

Figura 24: Creazione del gemello digitale

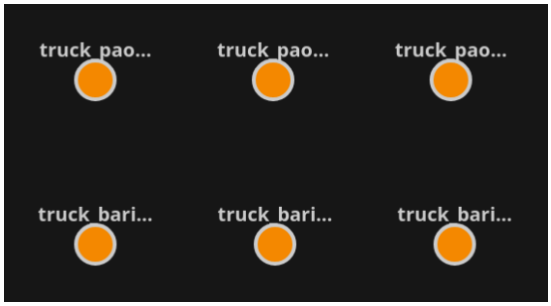

Figura 25: Visualizzazione dei gemelli digitali creati utilizzando il software Azure Digital Twin Explorer

4.2.7 Smart contract

Lo smart contract con funzione di notarizzazione è stato da prima costruito come si vede nel listato 2.

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract Hash {

    event HashSigned(bytes32 indexed hash, address indexed signer,
    ↪  SignData data);

    function initialize() public {}

    function signHash(bytes32 hash, string memory data) public {
        require(hash != bytes32(0), "Hash cannot be zero");
        require(!keyExists[hash], "Hash already signed");
        emit HashSigned(hash, msg.sender, signData[hash]);
    }

    function isHashSigned(bytes32 hash) public view returns (bool) {
        return keyExists[hash];
    }
}
```

Listing 2: Contratto di notarizzazione - versione iniziale

Durante la fase preliminare di scouting delle librerie sono stati individuati alcuni tool di interesse. Si è individuato il seguente stack tecnologico per la compilazione e la distribuzione di contratti scritti in Solidity ed eseguibili all'interno di una Ethereum Virtual Machine.

- **Solidity** scrittura del contratto
- **HardHat.js** compilazione, distribuzione e test dei contratti
- **Ether.js** client Ethereum per NodeJS

[H]

```javascript
import '@nomicfoundation/hardhat-toolbox';
import '@nomicfoundation/hardhat-verify';
import '@nomicfoundation/hardhat-ethers';
import '@openzeppelin/hardhat-upgrades';
import { task } from 'hardhat/config';
require('dotenv').config();
```

```
7
8  const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY as
   ↪  string;
9  const POLYSCAN_API_KEY = process.env.POLYSCAN_API_KEY as string;
10 const INFURA_API_KEY = process.env.INFURA_API_KEY as string;
11 const COINMARKETCAP_API_KEY = process.env.COINMARKETCAP_API_KEY
   ↪  as string;
12
13 const ACCOUNT_PVT_KEY = process.env.ACCOUNT_PVT_KEY as string;
14
15 const config = {
16   solidity: {
17     version: '0.8.24',
18     settings: {
19       optimizer: {
20          enabled: true,
21          runs: 1000
22       }
23     }
24   },
25   networks: {
26     polygonAmoy: {
27       url: `https://polygon-amoy-bor-rpc.publicnode.com`,
28       accounts: [ACCOUNT_PVT_KEY]
29     }
30   },
31   etherscan: {
32     apiKey: {
33       mainnet: ETHERSCAN_API_KEY,
34       holesky: ETHERSCAN_API_KEY,
35       polygonAmoy: POLYSCAN_API_KEY
36     }
37   },
38   sourcify: {
39     enabled: false
40   },
```

```
  gasReporter: {
    enabled: true,
    currency: 'EUR',
    darkMode: true,
    L1: 'polygon',
    currencyDisplayPrecision: 4,
    includeIntrinsicGas: true,
    coinmarketcap: COINMARKETCAP_API_KEY
  }
};

export default config;
```

Figura 26: HardHat Configurazione

[H]

```
import hre from 'hardhat';
import sha256 from 'crypto-js/sha256';
import {enc} from 'crypto-js';
import {Hash} from '../typechain-types';

require('dotenv').config();

async function main() {
  const Hash = await hre.ethers.getContractFactory('Hash');
  console.log('Sign an hash for message');
  const proxyAddress = process.env.HASH_PROXY_ADDRESS as string;
  // @ts-ignore
  const hash: Hash = Hash.attach(proxyAddress);

  let msg = JSON.stringify({
    data: 'This is my data! ' + (Math.floor(Math.random() * 9999)
    ↪  + 1000),
    timestampInternal: new Date().toLocaleString( 'sv', {
    ↪  timeZoneName: 'short' } )
  });
  const hexHash = sha256(msg);
```

```
20    const hashForSolidity = '0x' + hexHash.toString(enc.Hex);
21    console.log(hashForSolidity);
22    await hash.signHash(hashForSolidity, msg);
23    try {
24    } catch (err) {
25      console.log(`${hashForSolidity} is alredy signed`);
26    }
27    let check = await hash.isHashSigned(hashForSolidity);
28    console.log('is signed? ', check);
29    console.log('when? ', await getTimestampForSign(hash,
      ↪  hashForSolidity));
30    await getContractEvents(hash);
31  }
32
33  async function getContractEvents(contracts: Hash) {
34    const currentBlock: any = await
      ↪  hre.ethers.provider.getBlockNumber();
35    const eventFilter = contracts.filters.HashSigned();
36    const events = await contracts.queryFilter(eventFilter,
      ↪  currentBlock - 5000, currentBlock);
37    console.log(`Ci sono ${events.length} eventi di firma hash`);
38    for (let event of events) {
39      let block = await
        ↪  hre.ethers.provider.getBlock(event.blockNumber);
40      let data = await getData(contracts, event.args[0]);
41      // @ts-ignore
42      console.log(`${getDateFromTimeStamp(block.timestamp)} - Hash
        ↪  signed: ${JSON.stringify(event.args[0])} - Data Signed:
        ↪  ${data[2]}`);
43    }
44  }
45
46  function getDateFromTimeStamp(ts: any) {
47    let time: number = Number(ts) * 1000;
48    return new Date(time).toLocaleString( 'sv', { timeZoneName:
      ↪  'short' } );
```

```
}

async function getTimestampForSign(contract: any, hashString:
 ↪  any) {
  try {
    let hashData = await contract.getSignData(hashString);
    return getDateFromTimeStamp(hashData[1]);
  } catch (err) {
    return null;
  }
}

async function getData(contract: any, hashString: any) {
  try {
    return await contract.getSignData(hashString);
  } catch (err) {
    return null;
  }
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Figura 27: Script per la distribuzione del contratto Hash

[H]

```

```

Figura 28: Script per l'utilizzo del contratto Hash - Observer

[H]

```
import hre from 'hardhat';

```

```
async function main() {
  const Hash = await hre.ethers.getContractFactory('Hash');
  console.log('Deploying Hash...');
  const hash = await hre.upgrades.deployProxy(Hash);
  await hash.waitForDeployment();
  console.log('Hash deployed to:', await hash.getAddress());
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Figura 29: Script per l'utilizzo del contratto Hash - Signer

4.3 Implementazione - interfaccia

5 Risultati e conclusioni

6 Riferimenti

Elenco delle figure

Elenco delle tabelle

Elenco del codice sorgente

Riferimenti bibliografici

[1] *Angular — angular.dev*. https://angular.dev/overview. [Accessed 06-08-2024].

[2] *Core Libraries — VirtualThread — docs.oracle.com*. https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html. [Accessed 07-08-2024].

[3] *Documentation — NestJS - A progressive Node.js framework — docs.nestjs.com*. https://docs.nestjs.com/. [Accessed 06-08-2024].

[4] *Express - Framework per applicazioni web Node.js — expressjs.com*. https://expressjs.com/it/. [Accessed 06-08-2024].

[5] *Getting started with Hardhat — Ethereum development environment for professionals by Nomic Foundation — hardhat.org*. https://hardhat.org/hardhat-runner/docs/getting-started. [Accessed 06-08-2024].

[6] Lokesh Gupta. *REST Architectural Constraints — restfulapi.net*. https://restfulapi.net/rest-architectural-constraints/. [Accessed 06-08-2024].

[7] Dick Hardt. *RFC 6749: The OAuth 2.0 Authorization Framework — datatracker.ietf.org*. https://datatracker.ietf.org/doc/html/rfc6749. [Accessed 06-08-2024].

[8] *Hibernate ORM User Guide — docs.jboss.org*. https://docs.jboss.org/hibernate/stable/orm/userguide/html_single/Hibernate_User_Guide.html. [Accessed 06-08-2024].

[9] *Hibernate ORM User Guide — docs.jboss.org*. https://docs.jboss.org/hibernate/stable/orm/userguide/html_single/Hibernate_User_Guide.html. [Accessed 06-08-2024].

[10] *Inversione del controllo - Wikipedia — it.wikipedia.org*. https://it.wikipedia.org/wiki/Inversione_del_controllo. [Accessed 06-08-2024].

[11] Web3 Labs. *Quickstart - Web3j — docs.web3j.io*. https://docs.web3j.io/4.8.7/quickstart/. [Accessed 06-08-2024].

[12] *Node.js — Don't Block the Event Loop (or the Worker Pool) — nodejs.org*. https://nodejs.org/en/learn/asynchronous-work/dont-block-the-event-loop. [Accessed 06-08-2024].

[13] *OAuth grant types — Web Security Academy — portswigger.net*. https://portswigger.net/web-security/oauth/grant-types. [Accessed 07-08-2024].

[14] *React (web framework) - Wikipedia — it.wikipedia.org*. https://it.wikipedia.org/wiki/React_(web_framework). [Accessed 06-08-2024].

[15] *Solidity documentation — docs.soliditylang.org*. https://docs.soliditylang.org/en/latest/. [Accessed 14-07-2024].

[16] *Spring Boot - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Spring_Boot. [Accessed 06-08-2024].

[17] *State of JavaScript 2023: Front-end Frameworks — 2023.stateofjs.com*. https://2023.stateofjs.com/en-US/libraries/front-end-frameworks/. [Accessed 06-08-2024].

[18] Keycloak Team. *Docker - Keycloak — keycloak.org*. https://www.keycloak.org/getting-started/getting-started-docker. [Accessed 07-08-2024].

[19] *UUID Data Type — mariadb.com*. https://mariadb.com/kb/en/uuid-data-type/. [Accessed 07-08-2024].

[20] *Vue.js — vuejs.org*. https://vuejs.org/guide/introduction.html. [Accessed 06-08-2024].

[21] *What is a container? — docs.docker.com*. https://docs.docker.com/guides/docker-concepts/the-basics/what-is-a-container. [Accessed 07-08-2024].

[22] *Working with Virtual Threads in Spring 6 — Baeldung — baeldung.com*. https://www.baeldung.com/spring-6-virtual-threads. [Accessed 07-08-2024].