

Università della Calabria
Dipartimento di Matematica e Informatica



Corso di Laurea Magistrale in Informatica

Tesi di Laurea

Iot@Chain
ASP - Blockchain - IoT

Relatore:
Prof. Mario Alviano

Candidata:
Paola Guarasci
Matricola 231847

Anno Accademico 2023/2024

Indice

1	Introduzione	2
2	Background	2
2.1	Digital Twin	2
2.2	Blockchain	2
2.3	ASP	3
2.4	Concetti di Tracciabilità e di Filiera	3
3	Analisi dello stato dell'arte della tecnologia blockchain applicata alle supplychain	3
4	Definizione di un caso d'uso	3
4.1	Progettazione	4
4.1.1	Analisi dei requisiti	10
4.1.2	Casi d'uso	11
4.1.3	Persistenza	28
4.1.4	Definizione del modello di dominio	29
4.1.5	Answer Set Programming	40
4.1.6	Gemello digitale	40
4.1.7	Smart contract e blockchain	40
4.1.8	Riflessioni sulla sicurezza degli smart contract	43
4.2	Implementazione - server side	45
4.2.1	Autenticazione	47
4.2.2	Application Programming Interface (API)	49
4.2.3	Persistenza	54
4.2.4	Programmazione Asincrona e thread virtuali	54
4.2.5	Answer Set Programming	56
4.2.6	Gemello digitale	60
4.2.7	Interazione con la blockchain e smart contract	67
4.3	Implementazione - interfaccia	75
5	Risultati e conclusioni	84
6	Riferimenti	85

Sommario

 Lorem ipsum...

1 Introduzione

2 Background

2.1 Digital Twin

2.2 Blockchain

La tecnologia dei registri distribuiti, distributed ledger tecnology o DLT, consente 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

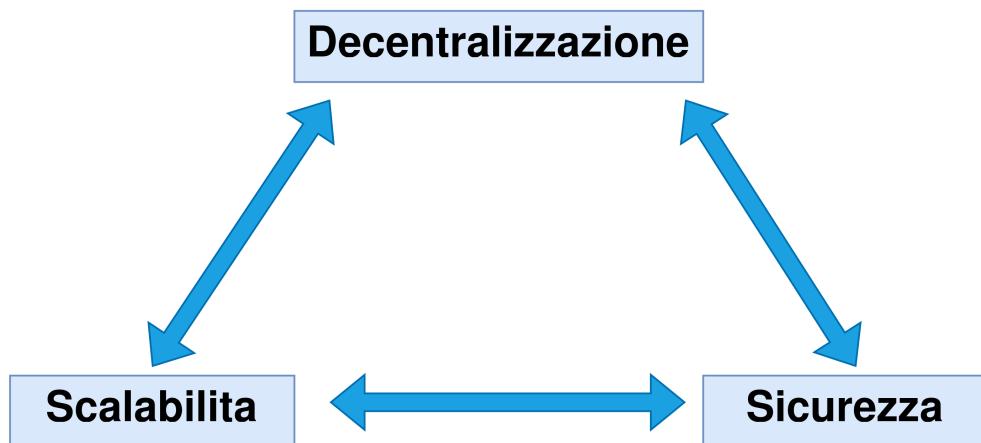


Figura 1: Il trilemma della blockchain

Il trilemma delle blockchain

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

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

Algoritmi di consenso

Permissionless 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 in un contesto di filiera. Il progetto generalizza molti aspetti della tracciabilità rendendolo di fatto agnostico rispetto al tipo di filiera che riesce a gestire. Nello specifico il prototipo tratta una filiera agroalimentare, ad esempio un prodotto surgelato di cui si vuole tenere traccia, in modo immutabile, dalla raccolta delle materie prime fino alla vendita finale. Si è deciso di implementare una piattaforma di gestione della filiera che integri gli aspetti tecnologici visti in precedenza: l'utilizzo dei gemelli digitali in un contesto di blockchain e che per qualche aspetto della logica di business risolva un quesito computazionale utilizzando l'answer set programming. Da qui in avanti sarà descritta la piattaforma software iotchain, analizzando la fase progettuale e poi dettagliando la fase implementativa. Descrivere minuziosamente ogni aspetto di un software che si è rivelato essere sufficiente complesso non è lo scopo di questa tesi, piuttosto ci si concentrerà di volta in volta sugli aspetti principali della progettazione e dell'implementazione con collegamenti diretti alle tecnologie richieste.

4.1 Progettazione

Il software che si intende sviluppare avrà un'architettura client-server, con un componente principale, il server, ed un componente secondario, il client, ed un'architettura 3-tier (3 livelli). I livelli di un'applicazione a tre tier sono:

- Livello di presentazione
- Livello applicazione (business logic)
- Livello dati

In questo caso il livello presentazione sarà stratificato, utilizzerà il pattern MVVM (Model-View-ViewModel [11]) in cui il *Model* è costituito da servizi che fanno da layer intermedi tra il livello presentazione e il livello applicazione. La comunicazione tra questi due livelli avviene tramite chiamate RESTAPI.



Figura 2: Architettura a livelli

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 architettonici [5]

- 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. [6]

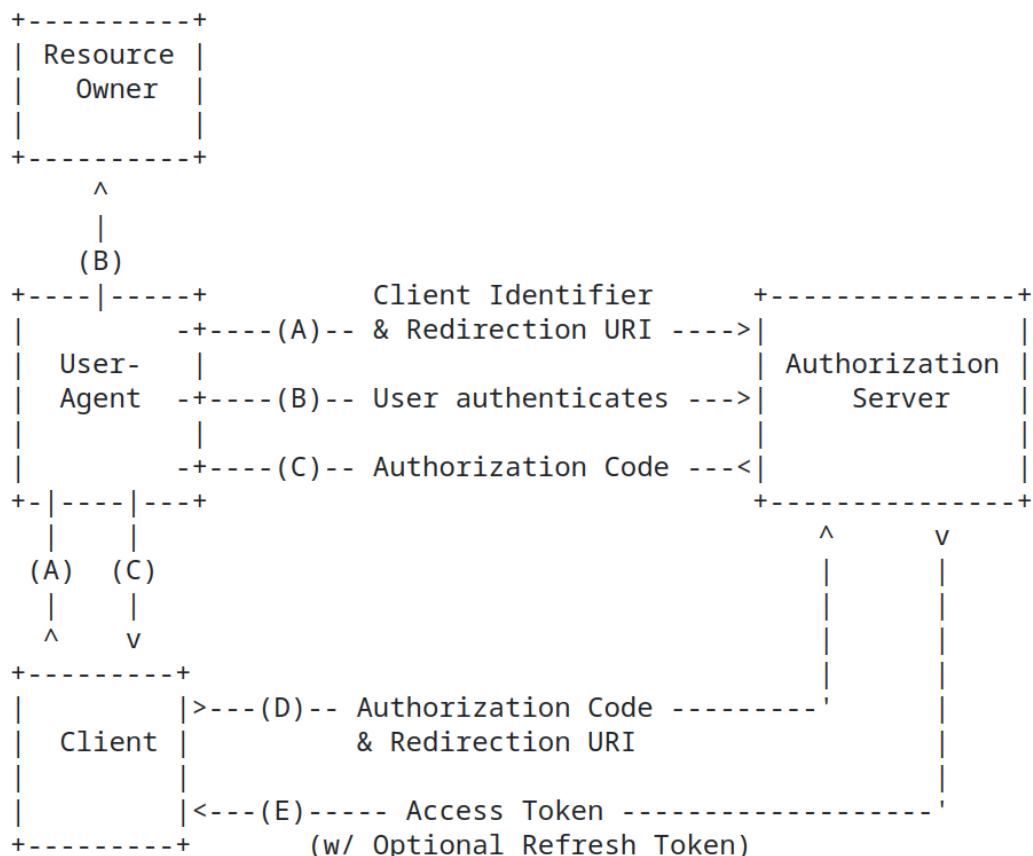


Figura 3: Authorization Flow in OAuth2 [6]

Lo stack tecnologico utilizzato comprende

Scelta dello stack tecnologico e considerazioni personali Sono stati presi in esame diversi framework per la componente server. In ambito NodeJS sono stati presi in considerazione NestJS ed Express, in ambiente Java invece è stato considerato Spring (SpringBoot).

NestJS è un framework che condivide con Spring alcune caratteristiche essenziali quali l'uso dei pattern di inversione del controllo e injection delle dipendenze. È costruito come livello aggiuntivo sopra framework più di basso livello, Express e Fastify, il cui utilizzo è esplicito e lo sviluppatore può decidere se adottare uno piuttosto che l'altro. Sia spring che nestjs, come anche Angular, utilizzano le annotazioni. NestJS condivide con Angular l'uso di Typescript, una versione tipizzata di javascript.

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. [3]

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, come già accennato, 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 procedurale 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. [9] 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 sviluppo. Esistono infatti 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. [15]

Dopo aver valutato le caratteristiche dei singoli framework, si è deciso che 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 sfruttare a pieno i vantaggi di una programmazione orientata

agli oggetti in un ambiente votato al multithread.

Altra caratteristica importante che ha fatto preferire spring rispetto agli altri framework presi in esame è sicuramente la gestione delle transazioni, globalmente ovvero lato applicativo ma anche tramite hibernate quindi sul livello della persistenza. Una transazione è un'operazione atomica che avviene all'interno di un'applicazione server o su una base di dati. Più in dettaglio le transazioni possono essere:

- Fisiche (transazioni della base di dati)
- Logiche (riferite ad un contesto generico di persistenza)
- Applicative (con riferimento al pattern architettonale Unit-of-Work)

In Hibernate le prime due vengono trattate come sinonimi. [7]. Spring Boot abilita in automatico la gestione delle transazioni ed è possibile utilizzare l'annotazione `@Transactional` su componenti di varia natura, tipicamente servizi. Tramite l'annotazione si possono configurare diversi aspetti della transazione:

- tipo di propagazione
- livello di isolamento
- timeout
- flag readonly
- regole di rollback

Le eccezioni che attivano un rollback sono eccezioni non gestite e che avvengono a tempo di esecuzione (`unchecked` e `runtime`). Si può forzare un comportamento diverso con alcuni parametri nell'annotazione. È possibile utilizzare tramite l'annotazione le transazioni del pacchetto `jakarta.transaction.Transactional` al posto delle transazioni di spring. [18]

Spring utilizza alcuni pattern:

- Singleton: ogni container di inversione del controllo o contesto applicativo esistono una sola istanza di una specifica classe
- Factory Method: sono alla base del concetto di iniezione delle dipendenze.
- Proxy: gli oggetti provenienti dalla persistenza quando caricati in maniera lazy vengono sostituiti da un oggetto proxy. L'oggetto reale è istanziato quando acceduto per la prima volta. Vengono usati anche per il controllo degli accessi nei metodi transazionali.

- Template Method: spring utilizza template per i contesti applicativi più comuni, ad esempio JDBC, JPA e transazioni.

Per l'implementazione della componenti client sono stati presi in esame i tre maggiori framework javascript ovvero 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 passa in prima posizione, Vue mantiene il secondo posto e React scivola in terza posizione [16].

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 (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. [13]

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]

La scelta è ricaduta su Angular per alcune sue caratteristiche interessanti: l'utilizzo delle annotazioni, l'uso di typescript e il 2way databinding.

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

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
- CU1a: Creazione di una ricetta (estende CU1 Registrazione di un tipo di prodotto)
- CU2: Registrazione di un lotto prodotto
- CU3: Tracciamento di un lotto (utente registrato)
- CU3a: Notarizzazione di uno step di processo (estende: CU3)

- CU4: Tracciamento di un lotto (utente non registrato)
- CU6x: Trasferimento
- CU5: Trasferimento di un lotto (atomico) [include CU6x]
- CU6: Trasferimento di un lotto (con accettazione) [include CU6x]
- 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	<ul style="list-style-type: none"> • 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" <ul style="list-style-type: none"> – Il sistema avvia il caso d'uso CU1a • Se l'utente clicca su salva <ul style="list-style-type: none"> – il sistema salva il nuovo tipo di prodotto • Se l'utente clicca su cancella <ul style="list-style-type: none"> – 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	<ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> – vai al passo 2 • Se l'utente clicca su rimuovi item <ul style="list-style-type: none"> – il sistema elimina l'item selezionato – vai al passo 1 • Se l'utente clicca su cancella <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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 salvata 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> 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: CU3 ⁴)	
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	<ul style="list-style-type: none"> 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 CU3 ⁴

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	<ul style="list-style-type: none"> L'utente è autenticato in piattaforma Esiste una compagnia B registrata in piattaforma La compagnia A ha un lotto di prodotto registrato
Sequenza	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Se C6x <ul style="list-style-type: none"> Il lotto creato è visibile alla compagnia destinataria Il lotto di partenza ha la quantità aggiornata Altrimenti <ul style="list-style-type: none"> 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	
ID	CU6x
Attori	–
Precondizioni	–
Sequenza	<ul style="list-style-type: none"> • 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 partire 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	<ul style="list-style-type: none"> • Utente A registrato che opera per conto di una compagnia A • Utente B registrato che opera per conto di una compagnia B
Precondizioni	<ul style="list-style-type: none"> • L'utente A è autenticato in piattaforma • L'utente B è autenticato in piattaforma • La compagnia A ha un lotto di prodotto registrato

Sequenza	<ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> – Il sistema avvisa l'utente • Se la quantità di prodotto è sufficiente <ul style="list-style-type: none"> – Il sistema blocca la risorsa da trasferire – Il sistema notifica all'utente B la presenza di un trasferimento in ingresso (caso 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 <ul style="list-style-type: none"> * Il sistema ripristina la quantità di prodotto bloccata nel lotto originale * Il caso d'uso termina
-----------------	--

Post-condizioni	<ul style="list-style-type: none"> • Se C6x 8 <ul style="list-style-type: none"> – Il lotto creato è visibile alla compagnia destinataria – Il lotto di partenza ha la quantità aggiornata • Altrimenti <ul style="list-style-type: none"> – 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Il caso d'uso inizia quando l'utente visualizza la lista delle transazioni • Se l'utente seleziona una transazione completata e clicca su "Invia" <ul style="list-style-type: none"> – Il sistema cerca un camion disponibile – Se ci sono camion disponibili <ul style="list-style-type: none"> * 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 <ul style="list-style-type: none"> * 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

Registrazione di un utente	
ID	CU12
Attori	Amministratore di sistema
Precondizioni	Utente autenticato in piattaforma
Sequenza	<ul style="list-style-type: none"> • Il caso d'uso inizia quando l'amministratore di sistema clicca su "aggiungi nuovo utente" • L'amministratore inserisce uno username, una password temporanea e un'azienda associta all'utente • L'amministratore clicca su "Salva" <ul style="list-style-type: none"> – Il sistema verifica che non ci siano altri utenti con il medesimo username <ul style="list-style-type: none"> * Se non sono presenti altri utenti con lo stesso username <ul style="list-style-type: none"> · Il sistema notifica all'amministratore l'impossibilità a procedere invitandolo a modificare il dato inserito · L'amministratore modifica lo username · L'amministratore clicca su salva e il caso d'uso riparete dal punto 3 * Se sono presenti altri utenti con lo stesso username <ul style="list-style-type: none"> · Il sistema salva il nuovo utente
Post-condizioni	L'utente è registrato in piattaforma e può effettuare il login (16)

Tabella 15: Caso d'uso 12 - Registrazione di un utente

Login di un utente	
ID	CU13
Attori	Utente registrato
Precondizioni	Utente registrato ma non autenticato in piattaforma
Sequenza	<ul style="list-style-type: none"> • Il caso d'uso inizia quando l'utente clicca su "login" • L'utente inserisce username e password • Il sistema verifica la correttezza dei dati inseriti dall'utente <ul style="list-style-type: none"> – Se i dati inseriti sono corretti ed è la prima volta che l'utente effettua il login <ul style="list-style-type: none"> * Il sistema impone il cambio password * L'utente inserisce una nuova password * Il sistema salva la nuova password * Il sistema autentica l'utente – Se i dati inseriti sono corretti e non è la prima volta che l'utente effettua il login <ul style="list-style-type: none"> * il sistema autentica l'utente – Se dati inseriti non sono corretti <ul style="list-style-type: none"> * il sistema notifica all'utente l'impossibilità a procedere
Post-condizioni	–

Tabella 16: Caso d'uso 13 - Login di un utente registrato

I casi d'uso esplicitati finqui si possono riassumere in un diagramma UML che li contiene tutti e schematizza le relazioni che intercorrono tra di essi.

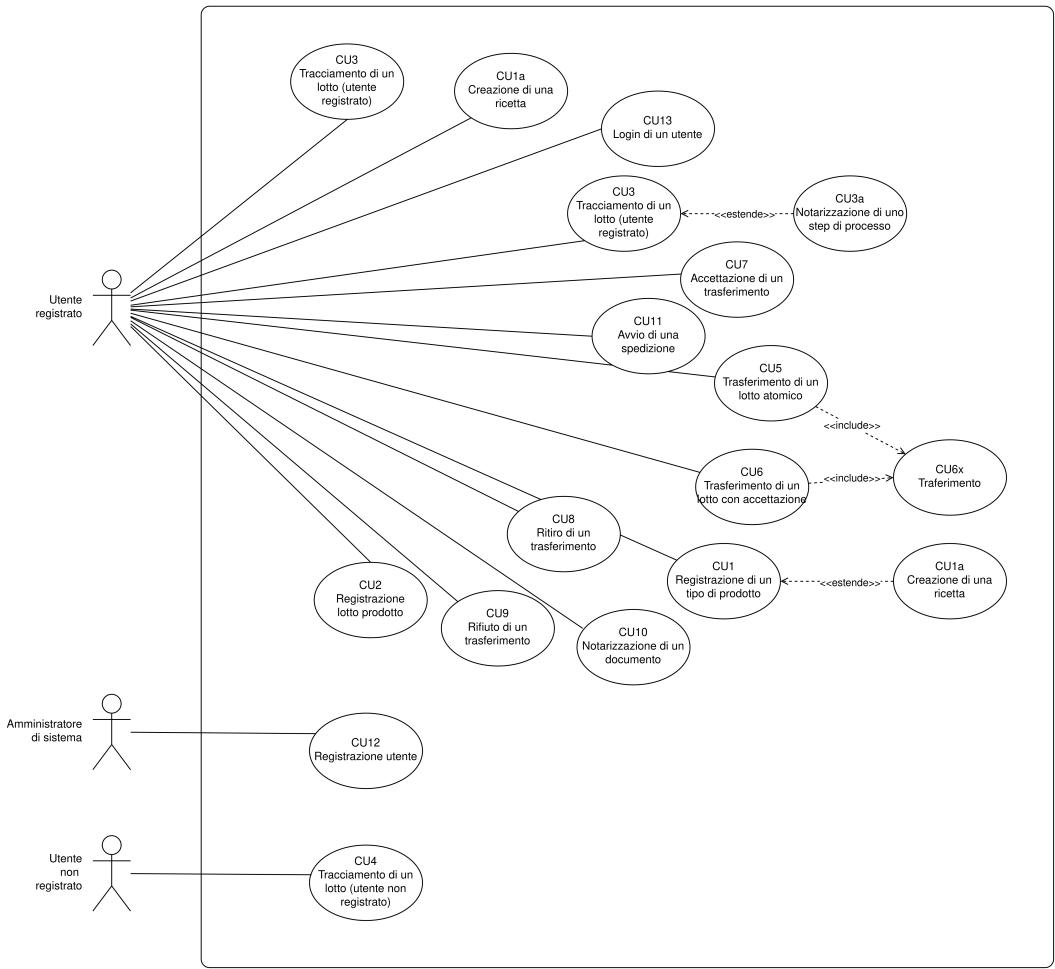


Figura 4: Diagramma UML dei casi d'uso

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 management 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]

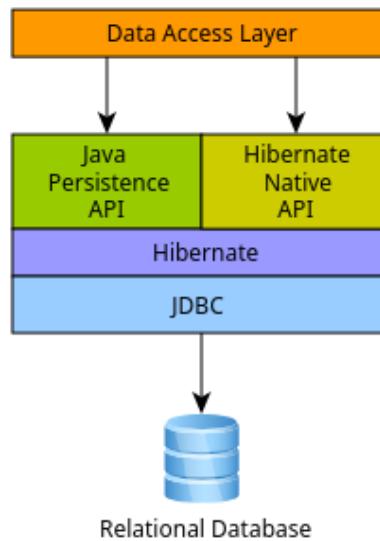


Figura 5: Dove si colloca Hibernate? [7]

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 un generico prodotto. È 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 può 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 può 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 obbligatoria
 - Descrizione: Ogni ricetta di tipo (Recipe) ha almeno un ingrediente di tipo(RecipeRow).
- RecipeBatch - RecipeRowBatch:
 - Cardinalità: uno a molti con partecipazione obbligatoria
 - 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

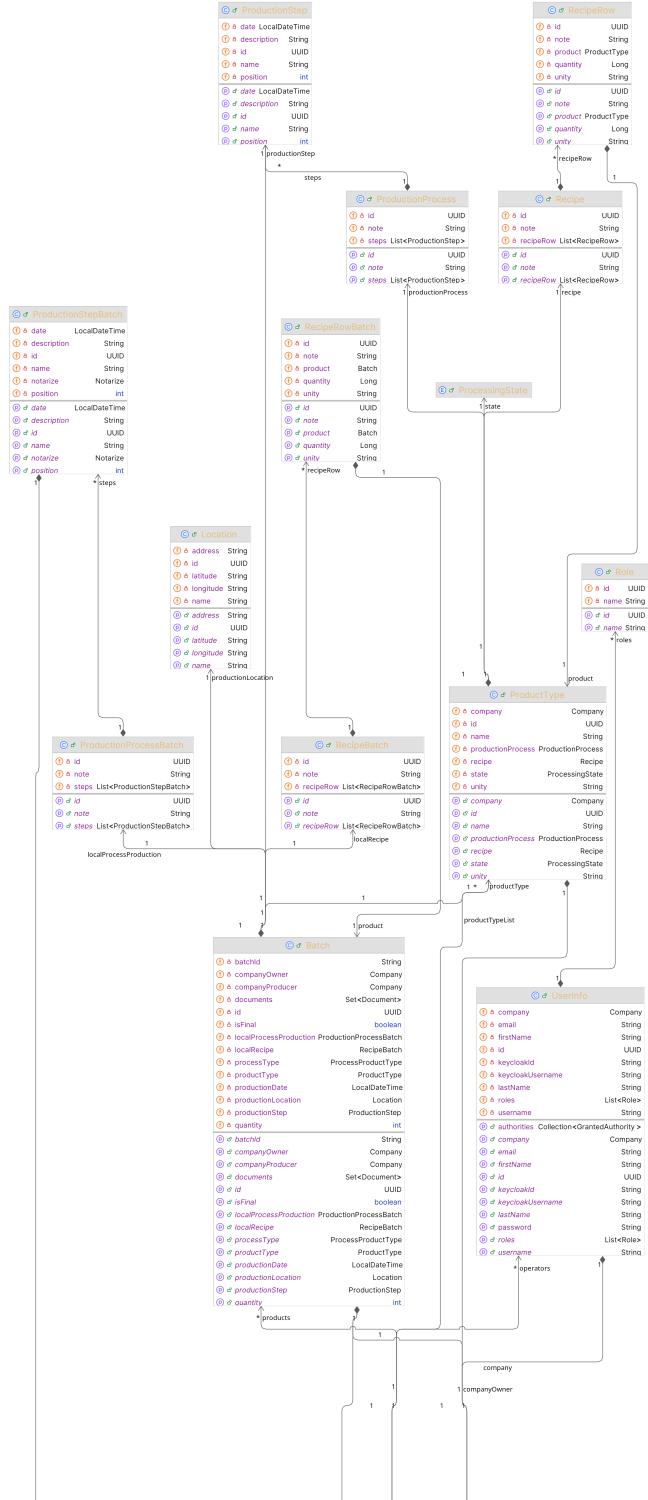


Figura 6: Diagramma UML classi - parte 1 di 2

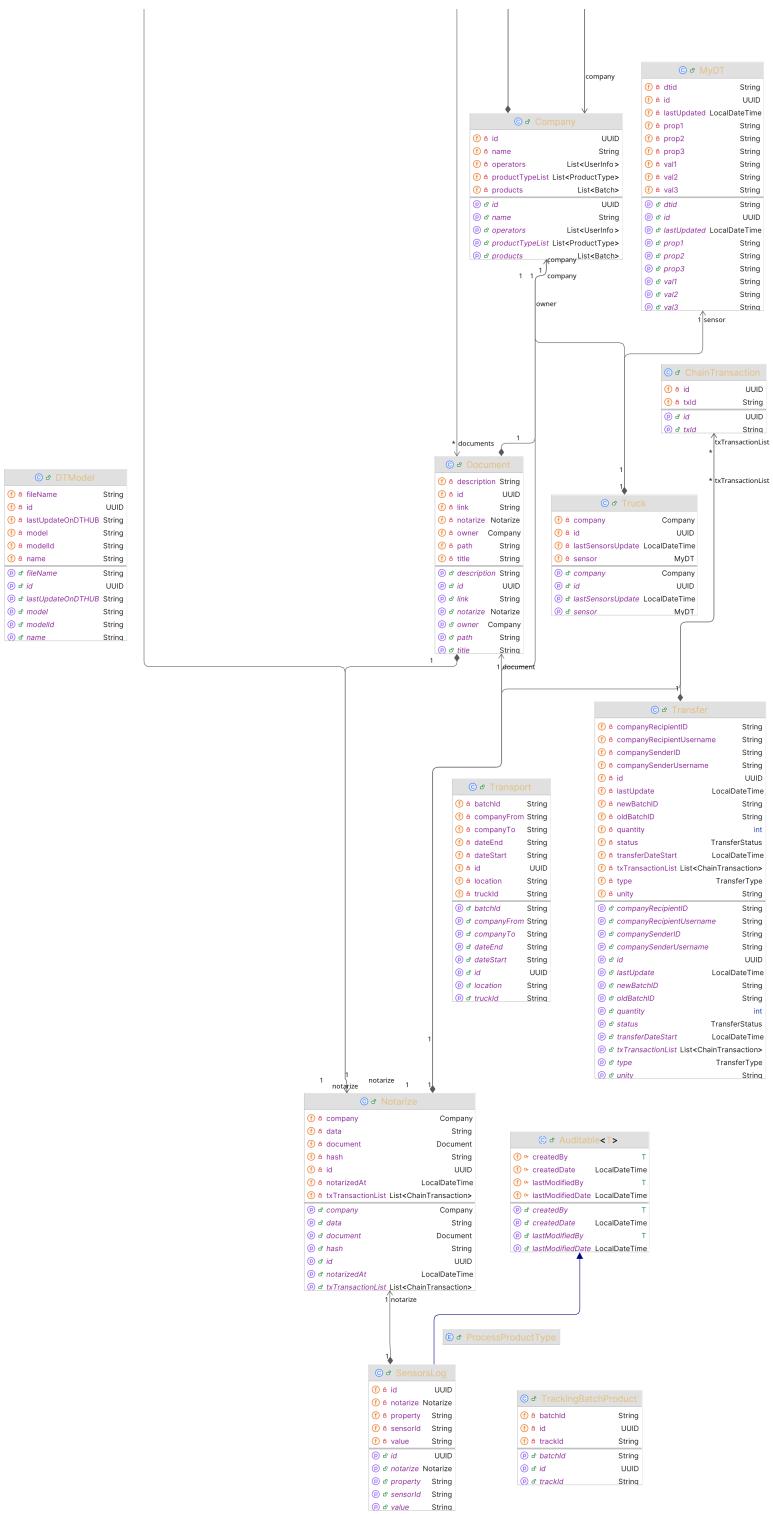


Figura 7: Diagramma UML classi - parte 2 di 2

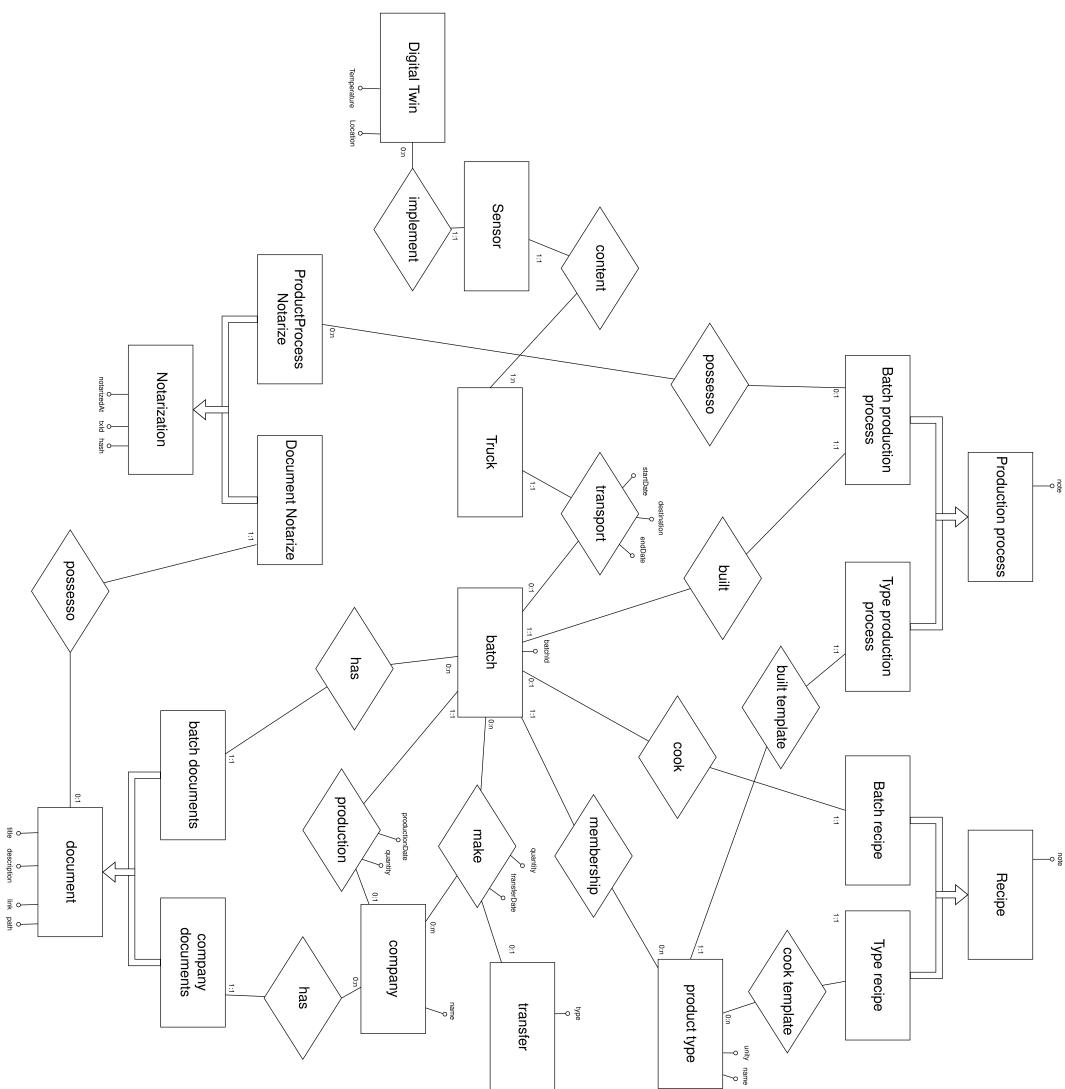


Figura 8: Schema Entità-Relazione

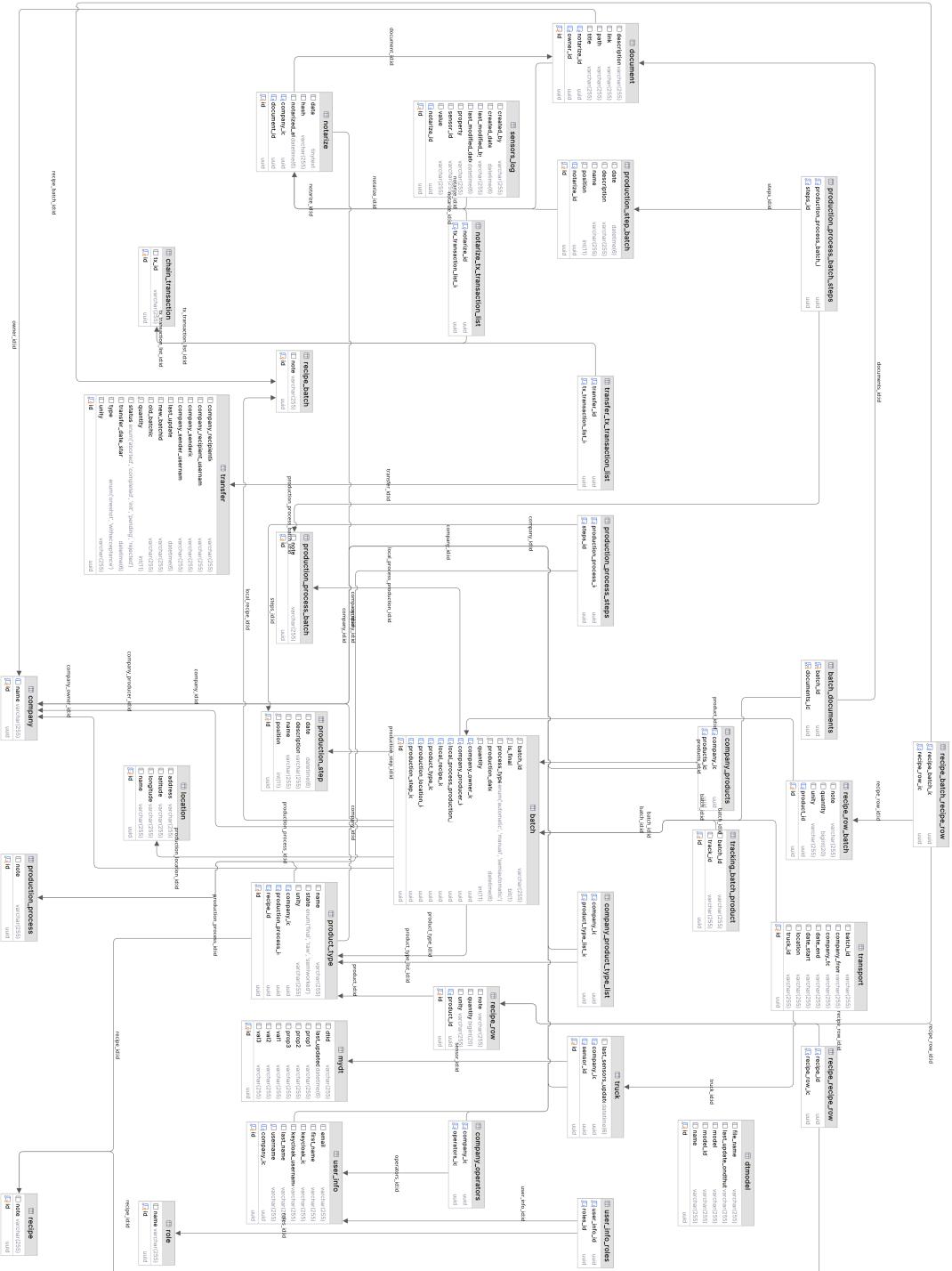


Figura 9: Schema logico base di dati

4.1.5 Answar Set Programming

Il modulo di answer set programming utilizza, tramite un wrapper non ufficiale rilasciato con licenza open source, il pacchetto clingo di Potassco 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 [14]

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** 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 tecnologico adottato è possibile inserire a questo proposito il progetto Web3j, una libreria java che supporta gli SmartContract e consente di interagire con la rete Ethereum tramite un client JSON-RPC, utilizzando chiamate API su protocollo HTTP o IPC. [10]

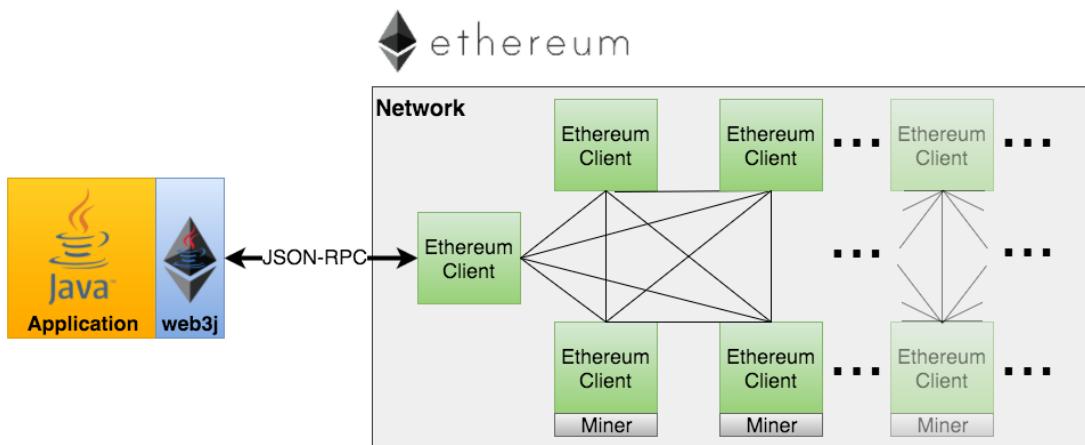


Figura 10: Web3j [10]

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

²Tradotto 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 differente 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. [4]

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/>

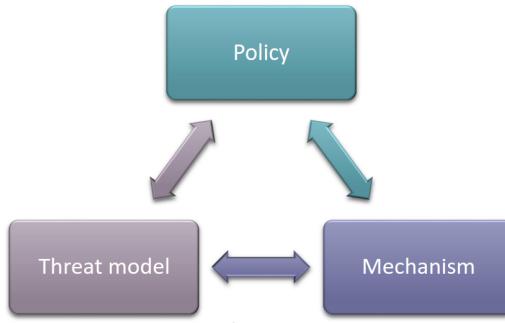


Figura 11: Elementi di sicurezza di un sistema informatico [slideMetodi]

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 Ownable 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

I componenti che incidono sugli aspetti di sicurezza di un sistema software possono essere divisi in 3 grandi macro aree: [slideMetodi]

- Meccanismi di sicurezza, quali autenticazione, autorizzazione o autid.
- Modello di minaccia, ovvero fare delle assunzioni sull’attaccante: che tipo di informazioni possiede e che poteri riesce ad ottenere. È importante che sia verosimile.
- Politica di gestione delle informazioni (integrità, confidenzialità, disponibilità, autenticità, non ripudio)

La modellazione di questi elementi e delle loro relazioni definisce la vulnerabilità di un sistema software.

In questo paragrafo vogliamo porre l’attenzione su di un particolare aspetto della piattaforma che si sta sviluppando, gli smart contract, e valutare le potenziali minacce e le possibili mitigazioni.

La documentazione ufficiale di Solidity elenca una serie di problematiche note [14] che possono dare vita a minacce di diversa natura, tra cui:

- Tutto quello che si usa in uno smart contract è pubblico
- L'utilizzo di numeri random è complesso se si vuole evitare la manomissione di questi numeri da parti di chi costruisce il blocco.
- Attacchi di reentrancy
- Gestione del gas (limiti e loop)
- Gestione dell'autenticazione in un contratto che fa uso dei proxy
- Overflow o underflow del complemento a due
- Campo `origin` della transazione usato per controllo ruoli o autenticazione
- Tipo mapping *sporco*

Anche OWASP fornisce dal 2023 una lista [**owaspOWASPSmart**] delle principali minacce di uno smart contract. In questa lista possiamo trovare ai primi posti:

- Attacco di reentrancy
- Overflow e underflow di interi
- Dipendenza dal timestamp
- Controllo degli accessi improprio

Un attacco di reentrancy sfrutta la vulnerabilità degli smart contract quando una funzione effettua una chiamata esterna a un altro contratto prima di aggiornare il proprio stato. Ciò consente al contratto esterno, eventualmente malevolo, di rientrare nella funzione originale e ripetere alcune azioni, come i prelievi, utilizzando lo stesso stato. Attraverso tali attacchi, un aggressore può eventualmente prosciugare tutti i fondi di un contratto. [**owaspOWASPSmart**]

Le mitigazioni suggerite sono:

- Always ensure that every state change happens before calling external contracts, i.e., update balances or code internally before calling external code. [**owaspReentrancy**]
- Use function modifiers that prevent reentrancy, like Open Zepplin's Reentrancy Guard. [**owaspReentrancy**]

Si è deciso per questo di utilizzare la guardia della libreria OpenZeppelin.

Un'altra potenziale vulnerabilità in questo contesto è un *Controllo degli accessi improprio*, definito come una vulnerabilità del controllo degli accessi è una falla nella sicurezza che consente a utenti non autorizzati di accedere o modificare i dati o le funzioni di un contratto. Queste vulnerabilità si verificano quando il codice del contratto non limita correttamente l'accesso in base alle autorizzazioni dell'utente. [owaspOWASPSmart] Le mitigazioni suggerite sono diverse tra cui l'uso del pattern Ownable nell'implementazione fornita da OpenZeppelin.

Pertanto la versione finale del contratto fa uso di Reentrancy Guard e Ownable, come si vede in figura

4.2 Implementazione - server side

Il caso d'uso implementato nel suo completto ha l'architettura che si vede in figura 12

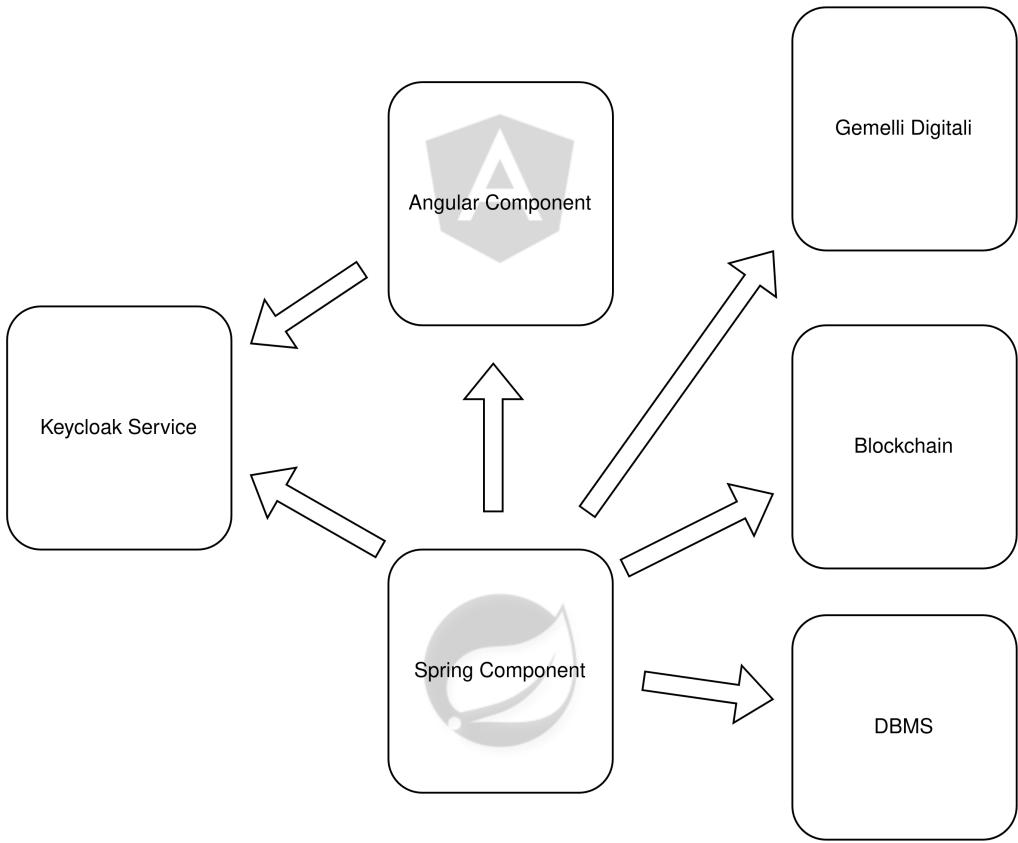


Figura 12: Architettura implementata

In questa sezione verranno analizzate le implementazioni più rilevanti. Sono state omesse descrizioni dettagliate di alcune componenti accessorie quali ad esempio le componenti responsabili della gestione delle operazioni CRUD⁵ di entità generiche. Sono state inserite le descrizioni delle medesime operazioni quando rilevanti nel contesto blockchain o digital twin.

La parte implementativa della componente principale, definita *server* o *componente spring*, inizia dalla creazione di un progetto di base utilizzando il tool online Spring Inizializer⁶.

⁵Acronimo che sta per Create, Read, Update, Delete. Rappresenta in breve la maggior parte delle operazioni effettuabili su di una specifica entità.

⁶Si tratta di un generatore di boilerplate ufficiale, disponibile gratuitamente all'indirizzo <https://start.spring.io/>

Successivamente si è proceduto alla configurazione del progetto e alla definizione delle variabili d'ambiente.

Dopo aver ottenuto un progetto Spring embrionale, si è avviata l'integrazione di servizi e componenti, tra cui ASP con clingo, la blockchain con la libreria Web3j e i gemelli digitali usando il client fornito dalle SDK di Microsoft Azure. Si è proceduto anche a configurare il servizio spring come resource server in un contesto Oauth2 e a configurare l'authentication server Keycloak.

I passi principali di queste integrazioni verranno spiegati di seguito.

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, Keycloak è l'authentication server, le risorse sono le informazioni veicolate tramite le chiamate alle API⁷ ed in fine la componente d'interfaccia, che viene eseguita nel browser dell'utente, è il client.

⁷Application Programming Interface - per una lista degli endpoints disponibili vedere ??

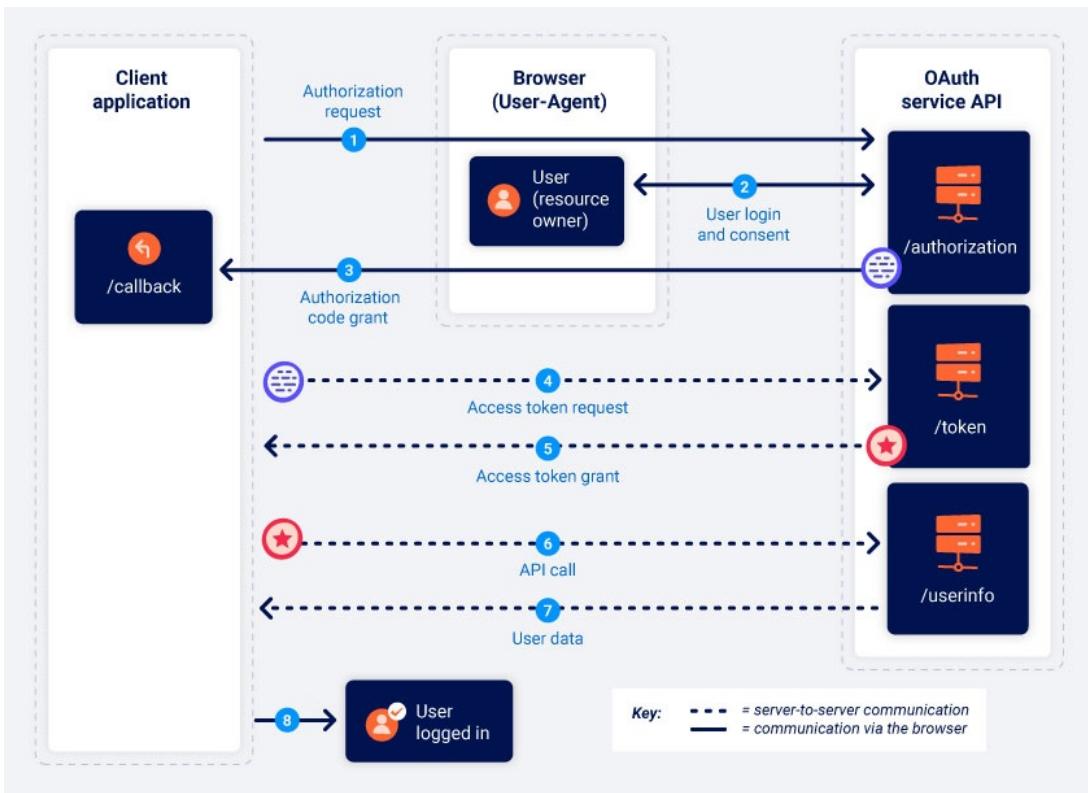


Figura 13: Authorization code grant type - da [12]

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]

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 [17]. L’adattamento effettuato serve a cari-

```

#!/usr/bin/bash
docker run -p 8881:8080 -e KEYCLOAK_ADMIN=admin -e
← KEYCLOAK_ADMIN_PASSWORD=admin -v ./data:/opt/keycloak/data/import
← quay.io/keycloak/keycloak:latest start-dev --import-realm

```

Listing 1: Keycloak - Comando di avvio

care 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: Endpoint che tiene conto dell'utente loggato e restituisce la lista di prodotti riferiti all'azienda per cui l'utente opera

Tipo di accesso: Utente registrato

Input: –

Output: Lista dei lotti di produzione della compagnia

- POST /api/v1/company/batch

Descrizione: Endpoint per la creazione di un lotto di prodotto

Tipo di accesso: Utente registrato

Input: Dati in formato json che rappresentano il lotto da creare

Output: Elemento appena creato

- GET /api/v1/company/batch/id

Descrizione: Endopoint per l'ottenimento di un singolo lotto di produzione.

Si effettua il check sull'azienda.

Tipo di accesso: Utente registrato

Input: id del lotto (uuid)

Output: rappresentazione per l'owner del lotto di produzione che risiede a livello persistenza

- GET /api/v1/company/batch/id/track
 Descrizione: informazioni di tracciamento del singolo lotto
 Tipo di accesso: Utente registrato
 Input: id del lotto (uuid)
 Output: informazioni aggregate in formato json sul lotto e sulle materie prime, riferimenti agli eventi di notarizzazione esistenti
- GET /api/v1/company/product-type
 Descrizione: lista dei tipi di prodotto aziendali
 Tipo di accesso: Utente registrato
 Input: –
 Output: lista in formato json di una proiezione per owner di tipi di prodotto aziendali
- POST /api/v1/company/product-type
 Descrizione: creazione del tipo di prodotto aziendale
 Tipo di accesso: Utente registrato
 Input: informazioni in formato json che definiscono il tipo di prodotto
 Output: elemento creato
- GET /api/v1/company/client
 Descrizione: Lista di aziende presenti in piattaforma con solo nome e id (è un endpoint di *servizio*)
 Tipo di accesso: Utente registrato
 Input: –
 Output: lista di elementi {id: "", name:""}
- GET /api/v1/company/transfer
 Descrizione: Lista di elementi che descrivono i trasferimenti avviati da un'azienda oppure trasferimenti per cui l'azienda è destinataria.
 Tipo di accesso: Utente registrato
 Input: –
 Output: lista trasferimenti (proiezione per owner)
- POST /api/v1/company/transfer
 Descrizione: Creazione di un trasferimento da una compagnia A ad una compagnia B. La compagnia A è sempre l'azienda legata all'utente che effettua la richiesta.
 Tipo di accesso: Utente registrato

Input: mappatura del trasferimento in formato json

Output: elemento creato

- GET /api/v1/company/transfer/{{batch_id}}

Descrizione: Lista di trasferimenti in cui è coinvolto un singolo lotto di produzione

Tipo di accesso: Utente registrato

Input: lotto di produzione

Output: lista elementi trasferimento

- GET /api/v1/company/transfer/{{trans_id}}/abort

Descrizione: Endpoint per la gestione dei trasferimenti con accettazione. Eliminazione del trasferimento da parte della compagnia che lo ha avviato. L'operazione va a buon fine solo se il trasferimento non è stato accettato/rifiutato dall'azienda destinataria.

Tipo di accesso: Utente registrato

Input: id trasferimento (UUID)

Output: messaggio di stato

- GET /api/v1/company/transfer/{{trans_id}}/accept

Descrizione: Endpoint per la gestione dei trasferimenti con accettazione. Accettazione di un trasferimento da parte della compagnia destinataria. L'operazione va a buon fine solo se l'utente che avvia la richiesta opera per conto della compagnia destinataria del trasferimento.

Tipo di accesso: Utente registrato

Input: id trasferimento (UUID)

Output: messaggio di stato

- GET /api/v1/company/transfer/{{trans_id}}/reject

Descrizione: Endpoint per la gestione dei trasferimenti con accettazione. Rifiuto del trasferimento da parte della compagnia destinataria. L'operazione va a buon fine solo se l'utente che avvia la richiesta opera per conto della compagnia destinataria del trasferimento.

Tipo di accesso: Utente registrato

Input: id trasferimento (UUID)

Output: messaggio di stato

- GET /api/v1/company/doc

Descrizione: Lista delle informazioni legate alle risorse presenti in piattaforma caricate dall'azienda, senza le risorse stesse. Per l'ottenimento dei

file va usato l'endpoint che consente il download di una risorsa singola.

Tipo di accesso: Utente registrato

Input: –

Output: lista in formato json delle informazioni presenti nella base di dati

- **GET /api/v1/company/doc/notarize/{doc_id}**

Descrizione: Endpoint che avvia la notarizzazione di uno specifico documento già presente in piattaforma

Tipo di accesso: Utente registrato

Input: id del documento (UUID)

Output: –

- **POST /api/v1/company/doc/upload**

Descrizione: Upload del documento in piattaforma, caricamento del file e delle informazioni aggiuntive.

Tipo di accesso: Utente registrato

Input: file in formato multipart

Output: elemento appena creato

- **GET /api/v1/company/doc/{doc_id}**

Descrizione: Ottenimento di un singolo elemento documentale (no binario)

Tipo di accesso: Utente registrato

Input: id del documento (UUID)

Output: json con le informazioni aggiuntive del file

- **GET /api/v1/company/doc/{doc_id}/resource**

Descrizione: download di una risorsa

Tipo di accesso: Utente registrato

Input: id del documento (UUID)

Output: generico binario `application/octet-stream`

- **GET /api/v1/company/transport**

Descrizione: Lista dei trasporti di azienda

Tipo di accesso: Utente registrato

Input: –

Output: lista proiezioni per owner

- **POST /api/v1/company/transport**

Descrizione: creazione di un trasporto

Tipo di accesso: Utente registrato

Input: proprietà del trasporto in formato json

Output: elemento appena creato

- GET /api/v1/company/transport/{batch_id}

Descrizione: Trasporto che riguarda uno specifico lotto di produzione

Tipo di accesso: Utente registrato

Input: batch id (id di lotto, no uuid)

Output: singola proiezione per owner

- GET /api/v1/company/transport/{transport_id}/truck

Descrizione: Elemento truck legato al trasporto con id passato come parametro dell'url. Controllo azienda owner del truck e utente che effettua la richiesta.

Tipo di accesso: Utente registrato

Input: id transport (UUID)

Output: proiezione per owner con dati sui sensori

- GET /api/v1/company/truck

Descrizione: Lista elementi truck di azienda

Tipo di accesso: Utente registrato

Input: –

Output: lista proiezioni per owner con dati sui sensori

- GET /api/v1/company/truck/update

Descrizione: Endpoint *di servizio* che attiva l'update dei gemelli digitali manualmente, prima che venga effettuato dallo scheduling di Spring.

Tipo di accesso: Utente registrato

Input: –

Output: –

- GET /api/v1/company/notarization

Descrizione: Lista elementi di notarizzazione dell'azienda dell'utente loggato.

Tipo di accesso: Utente registrato

Input: –

Output: lista proiezioni per owner

- POST /api/v1/company/notarization/step/{step_id}

Descrizione: Notarizzazione di un passaggio del processo di produzione.

Controlli su azienda e utente loggato.

Tipo di accesso: Utente registrato

Input: id step da notarizzare (UUID)

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 containerizzato).

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 (specificamente 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 possono 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. Offrono 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
// 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());
    };
}

```

Listing 14: Configurazione Asincrona da [22]

Una funzione annotata come `@Bean` consente di utilizzare 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 Answer Set Programming (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, al termine della computazione da parte del solutore ASP, effettuare una mappatura inversa, tokenizzando l'output in formato stringa e costruendo gli oggetti di interesse. Nella porzione di codice che segue si puo' vedere il programma in ASP che effettua un'ottimizzazione sulla ricerca dei lotti da inserire nella produzione di uno specifico lotto, nei termini di materie prime. Questo per evitare che l'utente debba attivamente e manualmente selezionare quanta parte prelevare da ogni lotto a sua disposizione (aziendale) per effettuare la produzione del lotto che si sta realizzando e per cui i lotti selezionati dal programma ASP costituiscono le materie prime.

```

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).

```

Listing 15: Programma ASP - Ricerca lotto - Esempio fatti in input

Il programma attualmente in uso nella piattaforma è il seguente IoTChain.

```

{ 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.

```

Listing 16: Programma ASP - Ricerca lotto

Analizzando nel dettaglio otteniamo, istruzione per istruzione:

- `scelto(E, Q) :- batch(E, X, Q), type(X).`

Il primo passaggio è effettuare una selezione dei lotti a disposizione filtrando per tipo. Quindi in scelto troviamo una proiezione dei batch del tipo presente in type. Da notare la presenza delle parentesi quadre che indica l'opzionalità della regola: non è detto che ogni elemento debba essere scelto. L'assenza di questo costrutto impedisce alla regola di ottimizzazione espressa dopo di entrare in funzione.

- `tot(T) :- T = #sum Q, E : scelto(E, Q) .`

Qui si sta utilizzando l'aggregato `sum` che somma gli elementi di un insieme, in questo caso l'insieme degli elementi Q nelle coppie (Q,E). È importante notare che l'aggregato opera su di un insieme pertanto se non si considera il valore Q nel contesto della tupla (Q,E), due valori identici di Q non parteciperanno al computo totale. L'aggregato `sum` nella sua forma generale è il seguente:

$$\#aggr\{e1;...;en\} < u$$

dove $e1, \dots, en$ sono elementi aggregati con $n \geq 0$,

$\#aggr \in \{\#\text{count}, \#\text{sum}, \#\text{max}, \#\text{min}\}$ è il nome di una funzione aggregato, $<\in \{<, \leq, =, , , >, \geq\}$ è una relazione e u è un termine. Dato un atomo aggregato a , l'espressione a e $!a$ sono letterali.[23] L'atomo $\text{tot}(T)$ rappresenta la quantità totale di risorse selezionate.

- `:= amount(A), tot(B), A > B.`

Vincolo forte (se non rispettato l'interpretazione non è un modello) per cui la valutazione è binaria, true/false, nella forma generica si ha

$$:- b1, \dots, bn. [w@l, t1, \dots, tm]$$

da [23]. Il vincolo esprime la necessità di avere una quantità totale per i lotti selezionati che non sia inferiore alla quantità richiesta.

- `#minimize Q, Q : scelto(E,Q) .`

Shorthand che indica una serie di vincoli deboli [23] per cui si tende ad ottimizzare, minimizzando in questo caso, un peso per ogni termine scelto presente nel modello. L'obiettivo in questo caso è di utilizzare il minor numero possibile di lotti. La penalità è proporzionale alla quantità di materia selezionata.

- `#show scelto/2.`

Filtro degli atomi in output, mostra solo il termine show con arită 2.

La funzione di mappatura in ingresso è specifica per atomi di una determinata arită. In questo contesto gli atomi hanno tutti arită non superiore a 3 quindi si è deciso di procedere con funzioni specifiche per arită 0, 1, 2 e 3. Qui è riportata a titolo esemplificativo la funzione che gestisce arită 3.

```
@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(",").append(
        " ");
    }
}
```

```

    return sb.toString();
}

```

Listing 17: ASP - Mapping in ingresso

Similmente in output si è scelto di differenziare gli atomi in base all'arità degli stessi, creando una funzione di selezione che tenga conto del nome dell'atomo e della sua arità.

```

@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;
}

```

Listing 18: ASP - Mapping in uscita

Le funzioni di mappatura dei risultati restituiscono in output liste di Pair o Triplet. Sono elementi che formalizzano tuple gestiti dalla libreria JavaTuples⁸.

Infine il solutore, che a basso livello effettua una chiamata al tool clingo, in un primo passaggio esegue il grounding del problema. L'output della fase di grounding, un programma senza variabili, costituisce poi l'input del solutore vero e proprio.

```

@Override
public List<String> solveFileAndString(String filename, String
    customString) throws IOException, URISyntaxException {

```

⁸<https://www.javatuples.org/apidocs/org/javatuples/Triplet.html>

```

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;
}

```

Listing 19: ASP - Solutore

4.2.6 Gemello digitale

La piattaforma costruita utilizza il concetto di gemello digitale per modellare dei camion virtuali all'interno di un sistema di tracciamento. Il gemello digitale, definito usando Digital Twins Definition Language (DTDL)⁹, è il seguente

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

```

⁹<https://azure.github.io/opendigitaltwins-dtdl/DTDL/v3/DTDL.v3.html>

```

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

```

Listing 20: Modello di Gemello Digitale del camion virtuale

Poiché si sta compiendo un’operazione di astrazione del camion interessano, in questo scenario, solo alcune specifiche proprietà:

- Temperatura (Temperature)
- Posizione (Location)

La creazione di un gemello digitale avviene ogni qualvolta si dispone la creazione di un elemento Truck (camion) in piattaforma, secondo una business logic che dal punto di vista dei gemelli digitali può essere considerata una scatola nera. Quindi la creazione dei gemelli digitali avviene dinamicamente attraverso l’utilizzo del client sincrono del servizio Digital Twin fornito dall’SDK di Microsoft Azure. Si instanzia un generico elemento di tipo DigitalTwin gli si assegna un modello inizialmente caricato sulla piattaforma Azure e lo si configura con le proprietà volute.

```

@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;
}

```

Listing 21: Creazione del gemello digitale

L'instanziazione del client utilizzato avviene nel costruttore del servizio e fa uso di credenziali di sistema, attraverso l'ottenimento di un token dal servizio Identità di Azure (è una Active Directory) nell'ambiente locale in cui è eseguita l'applicazione. DefaultCredential esegue un controllo sulle variabili d'ambiente e arriva alla richiesta di login nel browser, passando per le impostazioni di diverse applicazioni tra cui visual studio code o il tool a riga dicomando di Azure.

```

import com.azure.core.credential.TokenCredential;
import com.azure.digitaltwins.core.DigitalTwinsClient;
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();
    }
}

```

Listing 22: Gemello digitale - init del client

Gli aggiornamenti dei gemelli digitali sono ottenuti con la funzione @Schedule di Spring che consente di eseguire una funzione ad intervalli regolari. Il

```

// 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();
    }
}

```

```

// 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);
}

```

Listing 23: Gemello digitale - Schediling e polling

Lo scheduling effettua una chiamata alla funzione di aggiornamento. La funzione di aggiornamento esegue una richiesta di update su tutti i gemelli digitali registrati in piattaforma e salva il dato aggiornato sulla base di dati, come item della tabella di log SensorLog ed invia anche una richiesta di notarizzazione dei dati. Questo è il passaggio che rende immutabili le informazioni provenienti dai sensori dei gemelli digitali.

```

@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());
}

if (res.get(sensor.getProp2()) != null) {

    ↪ sensor.setVal2(String.valueOf(res.get(sensor.getProp2())));
    sensorsLogs.add(SensorsLog.builder()
        .sensorId(sensor.getDtid())
        .property(sensor.getProp2())
        .value(sensor.getVal2())
        .build());
}

```

```

    if (res.get(sensor.getProp3()) != null) {

        ↪ sensor.setVal3(String.valueOf(res.get(sensor.getProp3())));
        sensorsLogs.add(SensorsLog.builder()
            .sensorId(sensor.getDtid())
            .property(sensor.getProp3())
            .value(sensor.getVal3())
            .build());
    }

    List<MyDT> sensorSaved = myDTRepository.saveAll(sensors);
    List<SensorsLog> saved =
        ↪ sensorsLogRepository.saveAll(sensorsLogs);
    try {
        if (!saved.isEmpty()) {
            notarizeService.notarize(saved);
        }
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    } catch (TransactionException e) {
        throw new RuntimeException(e);
    }
});
}

```

Listing 24: Gemello digitale - funzione di aggiornamento

Il software di gestione fornito da Microsoft Azure, Azure Digital Twin Explorer, offre diverse funzionalità tra cui anche una visualizzazione grafica dei gemelli digitali creati. In questa immagine si può osservare la presenza di sei camion registrati sulla piattaforma.

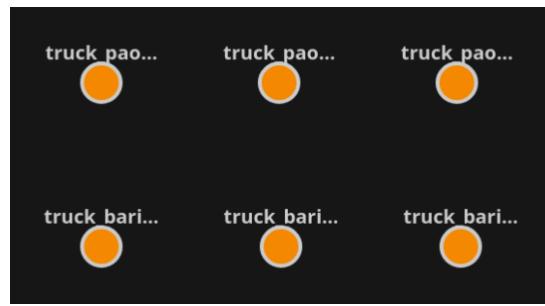


Figura 25: Visualizzazione dei gemelli digitali creati utilizzando il software Azure Digital Twin Explorer

4.2.7 Interazione con la blockchain e smart contract

Lo smart contract utilizzato ha una funzione di notarizzazione. Il contratto è stato da prima costruito come si vede nel listato 2.

```
// 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

In una fase successiva sono state apportate alcune evoluzioni. Si è deciso di implementare la classe Ownable di OpenZeppelin, per ragioni di sicurezza espresse

nel paragrafo dedicato [4.1.8](#), e si è deciso di distribuire il contratto utilizzando il concetto di proxy di OpenZeppelin. Una delle maggiori problematiche emerse durante lo sviluppo è stata l'impossibilità di avere un unico riferimento che rimanesse il medesimo anche a dispetto di numerose redistribuzioni del contratto stesso. I contratti, essendo immutabili per definizione, non si possono aggiornare e senza l'utilizzo dei proxy l'aggiornamento di un contratto equivale ad una nuova distribuzione che lascia comunque in essere la vecchia distribuzione, senza nessun riferimento ad essa. I proxy forniti da OpenZeppelin si frappongono tra l'utilizzatore e il contratto vero e proprio consentendo di avere sempre una versione aggiornata del contratto e utilizzando lo stesso indirizzo (quello del proxy). OpenZeppelin fornisce diverse implementazioni del concetto di proxy. Una di queste prevede semplicemente l'utilizzo della classe `Initializable`, che comporta l'eliminazione del costruttore e l'utilizzo di un metodo di inizializzazione. Questo perché i proxy non utilizzano il costruttore. ??

Quindi la versione finale del contratto ha questa forma:

```

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

// Import Ownable from the OpenZeppelin Contracts library
import "@openzeppelin/contracts/access/Ownable.sol";
import
    " @openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

contract Hash is Initializable {
    struct SignData {
        bytes32 hashSigned;
        uint256 timestamp;
        string data;
    }
    mapping(bytes32 => SignData) public signData;
    mapping(bytes32 => bool) public keyExists;

    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");

        keyExists[hash] = true;
        signData[hash] = SignData({
            hashSigned: hash,
            timestamp: block.timestamp,
            data: data
        });

        emit HashSigned(hash, msg.sender, signData[hash]);
    }

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

    function getSignData(
        bytes32 dataHash
    ) public view returns (bytes32, uint256, string memory) {
        SignData memory record = signData[dataHash];
        require(keyExists[dataHash], "Hash not signed");
        return (record.hashSigned, record.timestamp, record.data);
    }
}

```

69

Listing 3: Contratto di notarizzazione - versione finale

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

Hardhat.js ha come requisito minimo per poter funzionare la creazione di un file di configurazione all'interno della root di progetto chiamato `hardhat.config.ts`. La versione utilizzata in piattaforma è la seguente:

[H]

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

const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY as string;
const POLYSCAN_API_KEY = process.env.POLYSCAN_API_KEY as string;
const INFURA_API_KEY = process.env.INFURA_API_KEY as string;
const COINMARKETCAP_API_KEY = process.env.COINMARKETCAP_API_KEY as
↪ string;

const ACCOUNT_PVT_KEY = process.env.ACCOUNT_PVT_KEY as string;

const config = {
  solidity: {
    version: '0.8.24',
    settings: {
      optimizer: {
        enabled: true,
        runs: 1000
      }
    }
  }
}
```

```

        }
    },
    networks: {
        polygonAmoy: {
            url: `https://polygon-amoy-bor-rpc.publicnode.com`,
            accounts: [ACCOUNT_PVT_KEY]
        }
    },
    etherscan: {
        apiKey: {
            mainnet: ETHERSCAN_API_KEY,
            holesky: ETHERSCAN_API_KEY,
            polygonAmoy: POLYSCAN_API_KEY
        }
    },
    sourcify: {
        enabled: false
    },
    gasReporter: {
        enabled: true,
        currency: 'EUR',
        darkMode: true,
        L1: 'polygon',
        currencyDisplayPrecision: 4,
        includeIntrinsicGas: true,
        coinmarketcap: COINMARKETCAP_API_KEY
    }
};

export default config;

```

Listing 26: HardHat Configurazione

Per distribuire il contratto utilizzato, Hash, è stato necessario costruire un piccolo script che richiamasse le funzione di libreria atte allo scopo.

[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);
    const hashForSolidity = '0x' + hexHash.toString(enc.Hex);
    console.log(hashForSolidity);
    await hash.signHash(hashForSolidity, msg);
    try {
    } catch (err) {
        console.log(`#${hashForSolidity} is already signed`);
    }
    let check = await hash.isHashSigned(hashForSolidity);
    console.log('is signed? ', check);
    console.log('when? ', await getTimestampForSign(hash,
        hashForSolidity));
    await getContractEvents(hash);
}

async function getContractEvents(contracts: Hash) {

```

```

const currentBlock: any = await
  ↵ hre.ethers.provider.getBlockNumber();
const eventFilter = contracts.filters.HashSigned();
const events = await contracts.queryFilter(eventFilter,
  ↵ currentBlock - 5000, currentBlock);
console.log(`Ci sono ${events.length} eventi di firma hash`);
for (let event of events) {
  let block = await
    ↵ hre.ethers.provider.getBlock(event.blockNumber);
  let data = await getData(contracts, event.args[0]);
  // @ts-ignore
  console.log(` ${getDateFromTimeStamp(block.timestamp)} - Hash
    ↵ signed: ${JSON.stringify(event.args[0])} - Data Signed:
    ↵ ${data[2]}`);
}
}

function getDateFromTimeStamp(ts: any) {
  let time: number = Number(ts) * 1000;
  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);
});

```

Listing 27: Script per la distribuzione del contratto Hash

La fase di testing dei contratti riveste un ruolo fondamentale durante lo sviluppo degli stessi perché, come si è visto, i contratti sono immutabili e complessi da debuggare. Pertanto è conveniente utilizzare un approccio orientato al testing, come il Test Drive Development (TDD). Il contratto hash è stato largamente testato usando le librerie per tdd mocha e chai integrate in HardHat.

Solidity mette a disposizione gli eventi che sono definiti come un’astrazione del concetto di log sulle macchine virtuali etherium ¹⁰. A ben guardare però possono essere utilizzati con un pattern observer ed inseriti in un più ampio contesto di programmazione ad eventi. Per effettuare test di integrazione riguardo gli eventi e che coinvolgessero anche il modulo Spring sono stati utilizzati dei semplici script con ruoli di ascoltatori e scrittori. L’ascoltatore si pone in ascolto sugli eventi emessi dal contratto e ad ogni evento ricevuto compie un’azione triviale (effettua una stampa).

[H]

Listing 28: Script per l’utilizzo del contratto Hash - Observer

Lo script scrittore è leggermente più complesso perché effettua la scrittura sulla chain effettuando la chiamata alla funzione `signData`, avendo cura di codificare l’input nel formato atteso e successivamente viene decodificato l’output ricevuto.

[H]

¹⁰Documentazione ufficiale <https://docs.soliditylang.org/en/v0.8.26/contracts.html#events>

```

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);
});

```

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

4.3 Implementazione - interfaccia

Come anticipato nell'introduzione alla parte implementativa della componente server, anche in questo caso si è deciso di utilizzare un generatore di boilerplate, in questo caso si tratta del tool a riga di comando `ng` (Angular CLI), fornito dagli stessi sviluppatori del progetto principale. Successivamente si è proceduto alla configurazione personalizzata e all'implementazione di servizi che utilizzino le API fornite dalla componente server. I servizi costituiscono la componente *ponte* tra il livello applicazione e il livello presentazione e rappresentano la parte Model del pattern Model-View-ViewModel. La logica di visualizzazione e di aggiornamento della stessa risiede nei componenti di Angular che sfruttano il concetto di collegamento a due vie, 2way databinding. Il 2way databinding è la caratteristica di un componente di compiere delle azioni (tipicamente aggiornare la View) in risposta a degli eventi nel ViewModel. Un esempio triviale di 2way databinding può essere la visualizzazione di una variabile collegata ad un campo di input che riflette quello che l'utente sta digitando, in tempo reale e senza ricaricare la pagina.

Il passo finale è stata la modellazione di un interfaccia votata alla chiarezza e alla semplicità di utilizzo. Le librerie di componenti utilizzate sono:

- PrimeNG latest 17.18.8 ¹¹
- PrimeFlex 3.3.1 ¹²

In entrambi i casi si tratta di progetti con una versione denominata *community* e rilasciata con licenza MIT ¹³.

Nelle pagine che seguono sono stati inseriti delle immagini dalla piattaforma con spiegazioni dettagliate delle interfacce.

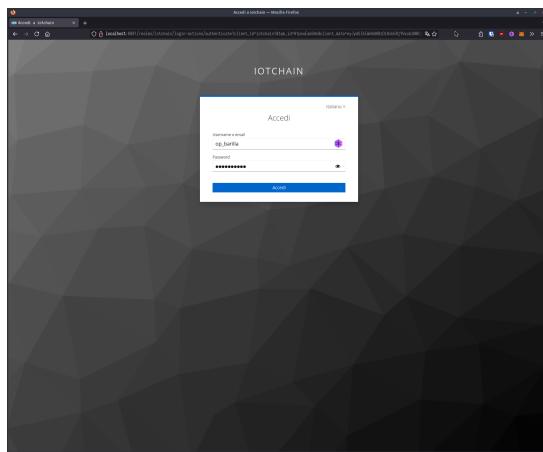


Figura 30: Pagina di login (fornita da keycloak)

Login La schermata di login prevede l'inserimento di uno username e di una password utilizzando la pagina web servita direttamente dal server Keycloak.

¹¹<https://primeng.org/>

¹²<https://primeflex.org/>

¹³https://en.wikipedia.org/wiki/MIT_License

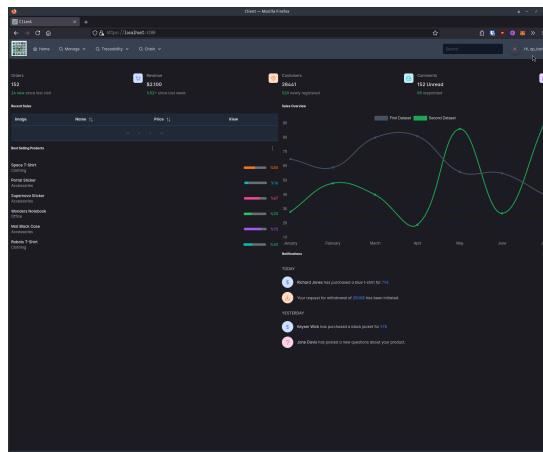


Figura 31: Dashboard (template)

Dashboard Dopo avere effettuato con successo il login, l’utente visualizza una dashboard contenente informazioni sintetiche e aggregate sullo stato della piattaforma dal punto di vista dell’azienda per cui opera. Ogni utente, è bene ricordarlo, non opera mai in prima persona ma opera sempre per conto dell’azienda per cui lavora. L’utente si muove all’interno della piattaforma utilizzando il menu situato nella barra superiore. Nella stessa barra, all’estrema destra dello schermo è possibile effettuare login/logout e sono visibili informazioni di base sull’identità dell’utente connesso (username e nome dell’azienda).

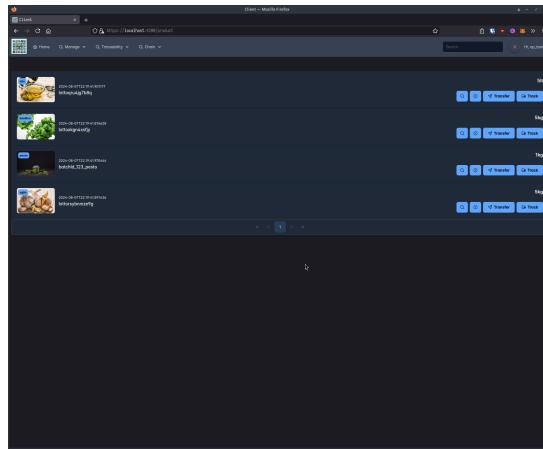


Figura 32: Lista dei lotti di produzione

Lista dei lotti di produzione Elenco per azienda dei lotti prodotti (dall'azienda) e registrati in piattaforma. Da questa visualizzazione è possibile accedere ad alcuni dettagli del lotto stesso:

- **click su (i)nformazioni** lista delle transazioni che riguardano le operazioni di scambio di uno specifico lotto (si veda Fig.33)
- **click su transfert** disporre il trasferimento del lotto o porzione di esso, in modalità atomica o con accettazione (si vedano Fig.?? e Fig.34)
- **click su track** informazioni di tracciamento del prodotto e delle sue materie prime, con eventuale indicazione di notarizzazione inserita (si veda Fig. 35)
- **click su lente di ingrandimento**

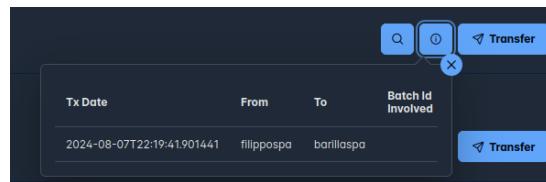


Figura 33: Operazioni di scambio che riguardano il lotto selezionato

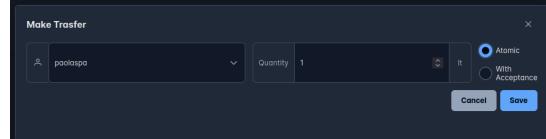


Figura 34: Disposizione di un trasferimento

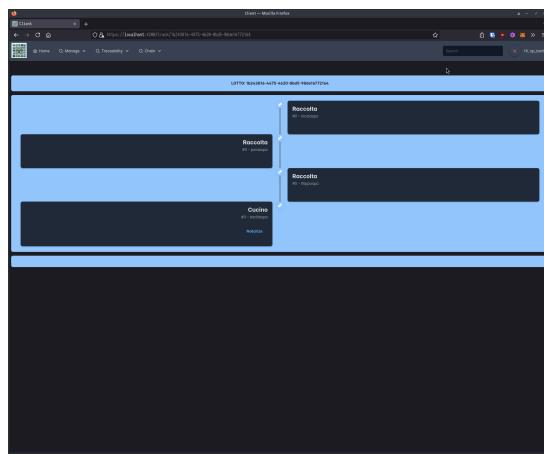


Figura 35: Visualizzazione di tracciamento di un lotto con track delle materie prime e operazioni di notarizzazione

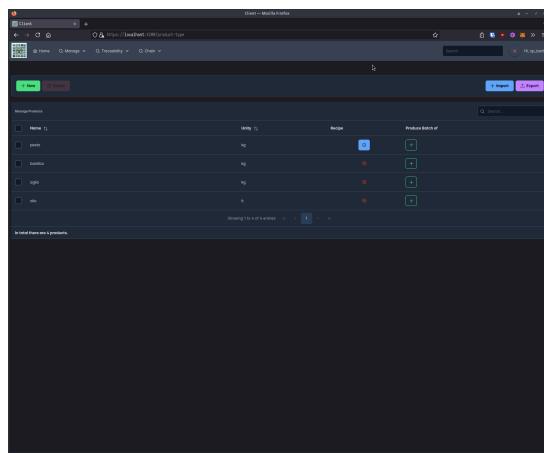


Figura 36: Lista tipi di prodotto

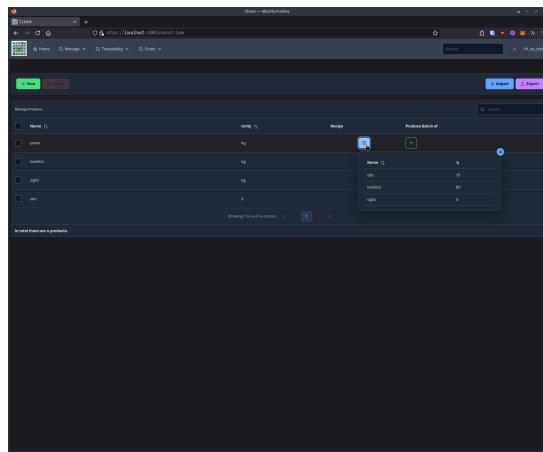


Figura 37: Lista tipi di prodotto - Particolare su ricetta di tipo

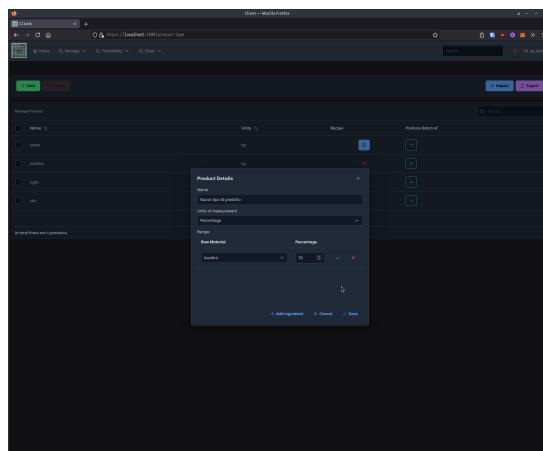


Figura 38: Creazione di un tipo di prodotto

Tipi di prodotto

Creazione di un lotto di prodotto In questa schermata è possibile inserire le informazioni necessarie alla creazione di un lotto di prodotto. I campi nome e descrizione sono campi di testo liberi. Anche se l'inserimento del testo è libero il campo nome verrà trasformato in modo tale da eliminare gli spazi e sostituirli con il carattere underscore. La selezione del tipo di prodotto comporta la modifica degli elementi Recipe e Step di produzione all'interno della pagina stessa quindi è consigliabile effettuare la selezione del tipo corretto in una fase iniziale della compilazione del form. Se si arriva in questa pagina dalla lista

di tipi prodotto, il modulo risulta già in parte compilato con il tipo selezionato. Quindi dopo aver deciso il tipo di prodotto per cui si sta creando una istanza di produzione si procede alla selezione di un sottoinsieme di elementi sia per quanto riguarda gli elementi della ricetta di tipo (che andranno a costituire la ricetta di prodotto) e sia alla selezione di un sottoinsieme di passi nel processo produttivo di cui si vuole tenere conto nelle fasi di tracciamento (che andranno a costituire il processo produttivo di lotto). Infine è possibile allegare uno o più documenti precedentemente caricati in piattaforma ed inserire una quantità espressa usando l'unità di misura del tipo di prodotto e definita in fase di creazione dello stesso. Dopo aver terminato la compilazione del form e cliccato su Salva il sistema restituisce una notifica di avvenuto salvataggio o di errore, a seconda dei casi.

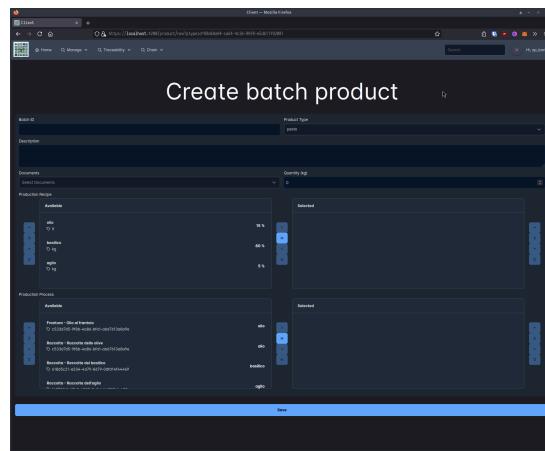


Figura 39: Creazione di un lotto di prodotto

Lista dei documenti In questa sezione della piattaforma è visibile una lista di elementi documentali che verosimilmente saranno certificati, analisi e altre attestazioni. Possono essere di prodotto o di azienda. Dopo aver caricato il file in uno dei formati accettati (pdf, jpg, png) è possibile procedere alla notarizzazione dello stesso e successivamente accedere alla pagina di dettaglio su cui sono riportate le informazioni della documentazione stessa. Ogni documento per poter essere allegato ad un lotto di produzione deve prima essere inserito in questa lista.

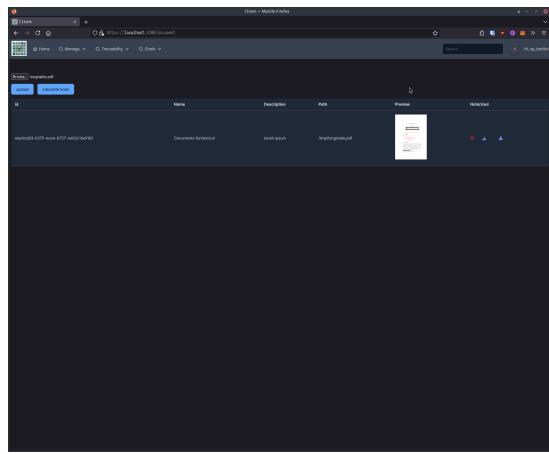


Figura 40: Lista di documenti caricati con anteprima

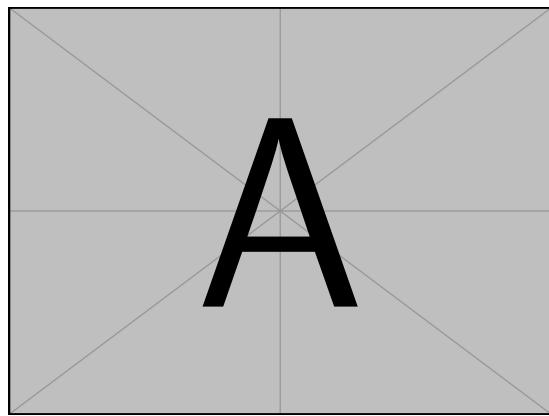


Figura 41: Informazioni di tracciamento e notarizzazione del documento

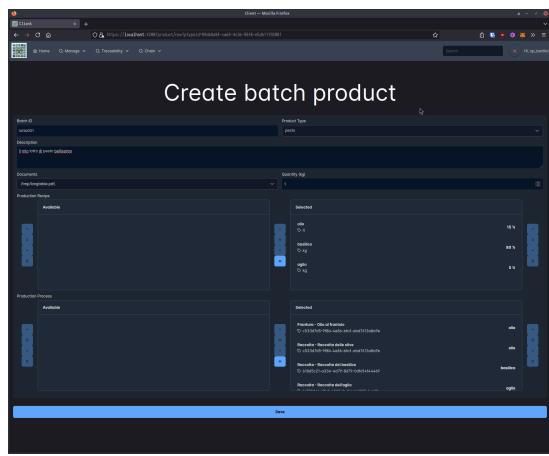


Figura 42: Lista di documenti caricati con anteprima

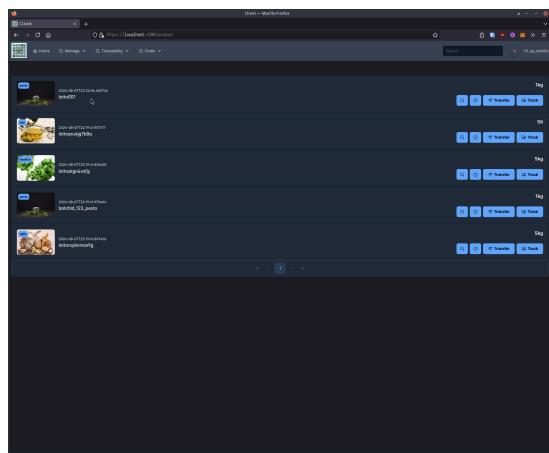


Figura 43: Lista di documenti caricati con anteprima

5 Risultati e conclusioni

6 Riferimenti

Elenco delle figure

1	Il trilemma della blockchain	2
2	Architettura a livelli	5
3	Authorization Flow in OAuth2 [6]	6
4	Diagramma UML dei casi d'uso	28
5	Dove si colloca Hibernate? [7]	29
6	Diagramma UML classi - parte 1 di 2	36
7	Diagramma UML classi - parte 2 di 2	37
8	Schema Entità-Relazione	38
9	Schema logico base di dati	39
10	Web3j [10]	41
11	Elementi di sicurezza di un sistema informatico [slideMetodi] . .	43
12	Architettura implementata	46
13	Authorization code grant type - da [12]	48
14	Configurazione Asincrona da [22]	56
15	Programma ASP - Ricerca lotto - Esempio fatti in input	57
16	Programma ASP - Ricerca lotto	57
17	ASP - Mapping in ingresso	59
18	ASP - Mapping in uscita	59
19	ASP - Solutore	60
20	Modello di Gemello Digitale del camion virtuale	61
21	Creazione del gemello digitale	62
22	Gemello digitale - init del client	62
23	Gemello digitale - Schediling e polling	64
24	Gemello digitale - funzione di aggiornamento	66

25	Visualizzazione dei gemelli digitali creati utilizzando il software Azure Digital Twin Explorer	67
26	HardHat Configurazione	71
27	Script per la distribuzione del contratto Hash	74
28	Script per l'utilizzo del contratto Hash - Observer	74
29	Script per l'utilizzo del contratto Hash - Signer	75
30	Pagina di login (fornita da keycloak)	76
31	Dashboard (template)	77
32	Lista dei lotti di produzione	77
33	Operazioni di scambio che riguardano il lotto selezionato	78
34	Disposizione di un trasferimento	78
35	Visualizzazione di tracciamento di un lotto con track delle materie prime e operazioni di notarizzazione	79
36	Lista tipi di prodotto	79
37	Lista tipi di prodotto - Particolare su ricetta di tipo	80
38	Creazione di un tipo di prodotto	80
39	Creazione di un lotto di prodotto	81
40	Lista di documenti caricati con anteprima	82
41	Informazioni di tracciamento e notarizzazione del documento	82
42	Lista di documenti caricati con anteprima	83
43	Lista di documenti caricati con anteprima	83

Elenco delle tabelle

1	Caso d'uso 1 - Registrazione di un tipo di prodotto	13
2	Caso d'uso 1a - Creazione di una ricetta (estende CU1 Registrazione di un tipo di prodotto)	14
3	Caso d'uso 2 - Registrazione di un lotto di prodotto	15
4	Caso d'uso 3 - Tracciamento di un lotto (utente registrato	16
5	Caso d'uso 3a - Notarizzazione di uno step di processo	16
6	Caso d'uso 4 - Tracciamento di un lotto (utente registrato	17
7	Caso d'uso 5 - Trasferimento di un lotto (atomico) [INCLUDE trasferimento]	18
8	Caso d'uso 6x - Trasferimento	19
9	Caso d'uso 6 - Trasferimento di un lotto con accettazione [INCLUDE C6x 8 Trasferimento]	21
10	Caso d'uso 7 - Accettazione di un trasferimento	21
11	Caso d'uso 8 - Ritiro di un trasferimento	22
12	Caso d'uso 9 - Rifiuto di un trasferimento	23
13	Caso d'uso 10 - Notarizzazione di un documento	24
14	Caso d'uso 11 - Avvio di una spedizione	25
15	Caso d'uso 12 - Registrazione di un utente	26
16	Caso d'uso 13 - Login di un utente registrato	27

Elenco del codice sorgente

1	Keycloak - Comando di avvio	49
2	Contratto di notarizzazione - versione iniziale	67
3	Contratto di notarizzazione - versione finale	69

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] *Express - Framework per applicazioni web Node.js* — expressjs.com. <https://expressjs.com/it/>. [Accessed 06-08-2024].
- [4] *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].
- [5] Lokesh Gupta. *REST Architectural Constraints* — restfulapi.net. <https://restfulapi.net/rest-architectural-constraints/>. [Accessed 06-08-2024].
- [6] Dick Hardt. *RFC 6749: The OAuth 2.0 Authorization Framework* — data-tracker.ietf.org. <https://datatracker.ietf.org/doc/html/rfc6749>. [Accessed 06-08-2024].
- [7] *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].
- [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] *Inversione del controllo* - Wikipedia — it.wikipedia.org. https://it.wikipedia.org/wiki/Inversione_del_controllo. [Accessed 06-08-2024].
- [10] Web3 Labs. *Quickstart - Web3j* — docs.web3j.io. <https://docs.web3j.io/4.8/quickstart/>. [Accessed 06-08-2024].
- [11] *Model-view-viewmodel* - Wikipedia — it.wikipedia.org. <https://it.wikipedia.org/wiki/Model-view-viewmodel>. [Accessed 09-08-2024].
- [12] *OAuth grant types* — Web Security Academy — portswigger.net. <https://portswigger.net/web-security/oauth/grant-types>. [Accessed 07-08-2024].
- [13] *React (web framework)* - Wikipedia — it.wikipedia.org. [https://it.wikipedia.org/wiki/React_\(web_framework\)](https://it.wikipedia.org/wiki/React_(web_framework)). [Accessed 06-08-2024].

- [14] *Solidity documentation* — docs.soliditylang.org. <https://docs.soliditylang.org/en/latest/>. [Accessed 14-07-2024].
- [15] *Spring Boot - Wikipedia* — en.wikipedia.org. https://en.wikipedia.org/wiki/Spring_Boot. [Accessed 06-08-2024].
- [16] *State of JavaScript 2023: Front-end Frameworks* — 2023.stateofjs.com. <https://2023.stateofjs.com/en-US/libraries/front-end-frameworks/>. [Accessed 06-08-2024].
- [17] Keycloak Team. *Docker - Keycloak* — keycloak.org. <https://www.keycloak.org/getting-started/getting-started-docker>. [Accessed 07-08-2024].
- [18] *Using @Transactional :: Spring Framework* — docs.spring.io. <https://docs.spring.io/spring-framework/reference/data-access/transaction/declarative/annotations.html>. [Accessed 09-08-2024].
- [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.com. <https://www.baeldung.com/spring-6-virtual-threads>. [Accessed 07-08-2024].
- [23] FRANCESCO CALIMERI et al. «ASP-Core-2 Input Language Format». In: *Theory and Practice of Logic Programming* 20.2 (dic. 2019), pp. 294–309. ISSN: 1475-3081. DOI: [10.1017/s1471068419000450](https://doi.org/10.1017/s1471068419000450). URL: <http://dx.doi.org/10.1017/S1471068419000450>.

Ringraziamenti