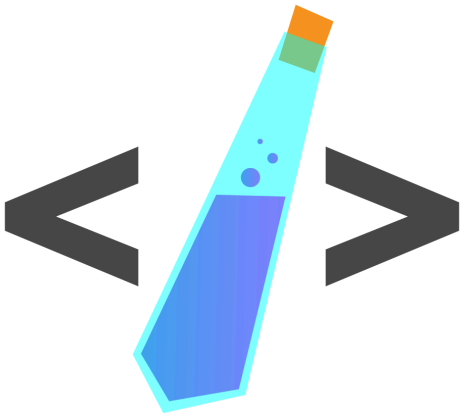

CODE ALCHEMISTS

TURNING IDEAS INTO CODE, LIKE MAGIC

Specifica Tecnica

Stato

Approvato

Versione

1.0.0

Distribuzione

Code Alchemists
M31
Prof. Tullio Vardanega
Prof. Riccardo Cardin

Registro delle Modifiche

Vers.	Data	Autore	Verificatore	Descrizione
1.0.0	15/09/2025	S. Marana A. Shu	N. Moretto	Revisione finale ed approvazione del documento
0.8.0	15/09/2025	A. Shu	N. Moretto	Completata specifica del Microservizio di Inventario Aggregato, Aggiornamento Microservizio di Ordini Aggregato e revisione generale del documento portando aggiornamento dei attributi e metodi mancanti nei microservizi già esistenti, aggiunta immagini mancanti
0.7.0	11/09/2025	S. Speranza	N. Moretto	Completata specifica del Microservizio di Autenticazione
0.6.1	08/09/2025	S. Speranza	N. Moretto	Iniziata specifica del Microservizio di Autenticazione
0.6.0	08/09/2025	S. Speranza	N. Moretto	Completata specifica del Microservizio Ordini
0.5.0	07/09/2025	S. Speranza	M. Dalla Pozza	Completata specifica del Microservizio Inventario
0.4.0	30/08/2025	A. Shu	M. Dalla Pozza	Aggiunta breve descrizioni sui microservizi (inventario, state, Cloud State, Sistema centralizzato e Routing)
0.3.0	28/08/2025	A. Shu N. Bolzon	M. Dalla Pozza	Aggiunta una documentazione basica del Microservizio Routing, correzioni granulari di stile e uniformità lessicale sulla stesura

Vers.	Data	Autore	Verificatore	Descrizione
0.2.0	27/08/2025	N. Bolzon A. Shu	M. Dalla Pozza	Stesura del documento, sezioni Introduzione, Tecnologie, Architettura.
0.1.0	09/08/2025	M. Dalla Pozza	S. Marana	Stesura del documento, sezioni Tecnologie utilizzate e architettura.
0.0.1	30/07/2025	N. Moretto	S. Speranza	Creazione template e struttura del documento.

Indice

Lista delle Immagini

Lista delle Tabelle

1. Introduzione

1.1. Scopo del documento

Il seguente documento ha lo scopo di descrivere dettagliatamente l'architettura del prodotto software considerato tramite una visione chiara e strutturata delle sue componenti, delle interazioni tra di essi e la distribuzione nel sistema.

È un documento tecnico che si rivolge principalmente agli sviluppatori e ai membri del team di progetto, ma può essere utile anche per altri stakeholder interessati a comprendere le basi tecniche del sistema.

In particolare, il documento si propone di:

- Fornire una descrizione dettagliata dell'architettura logica del sistema, inclusi i componenti principali, le loro interazioni e le tecnologie utilizzate;
- Descrivere le scelte progettuali effettuate e le motivazioni alla base di tali scelte;
- Presentare i design patterns utilizzati e le loro applicazioni nel contesto del progetto;
- Presentare lo stato dei requisiti funzionali implementati, evidenziando quelli completati e quelli ancora in fase di sviluppo;
- Fornire ulteriori dettagli tecnici e progettuali necessari per una migliore comprensione e la manutenzione del sistema;

1.2. Glossario

Per una corretta comprensione del documento, viene fornito un glossario dei termini utilizzati. Ogni termine è indicato da una «^{G}» in apice alla parola^{G}. Per trovare il significato del termine, è possibile consultare il glossario al seguente indirizzo: https://teamcodealchemists.github.io/glossario.html

1.3. Riferimenti

1.3.1. Riferimenti normativi

- **Capitolato^{G} d'appalto C6 - Gestione di un magazzino distribuito - *M31***
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C6.pdf
- **Norme di Progetto^{G} - versione 1.0.0**
  https://teamcodealchemists.github.io/docs/pb/NdP_1.0.0.pdf
  Ultimo Accesso: 27 agosto 2025

1.3.2. Riferimenti informativi

- **Lezione rovesciata - Documentazione**
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/FC1.pdf
  Ultimo Accesso: 1 agosto 2025
- **Regolamento del Progetto Didattico**
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf
  Ultimo Accesso: 1 agosto 2025

- **Glossario**
  https://teamcodealchemists.github.io/glossario.html
  Ultimo Accesso: 27 agosto 2025

2. Tecnologie

Questa sezione ha l'intento di offrire una panoramica delle tecnologie adottate nella progettazione del sistema software, esaminando con attenzione le piattaforme, gli strumenti, i linguaggi di programmazione, i framework e le risorse tecnologiche impiegate durante l'intero ciclo di sviluppo.

Le tecnologie proposte e descritte sono state selezionate dalla necessità di sviluppare un sistema centralizzato composto da magazzini distribuiti, capitolato di progetto C6[G], capace di operare in modo efficiente anche sotto carico, variabile e non, mantenendo prestazioni elevate e un'ottima resilienza.

Il progetto quindi si fonda su una selezione di strumenti avanzati e affidabili, scelti per le loro capacità di supportare in maniera efficiente un'architettura in particolare basata su microservizi, garantendo al contempo la scalabilità, l'affidabilità e una gestione semplificata della manutenzione.

Di seguito vengo presentate le tecnologie usate suddivise in categorie in base al loro ruolo all'interno dell'architettura:

- linguaggi di programmazione per lo sviluppo del codice
- strumenti per la comunicazione tra microservizi
- soluzioni per la virtualizzazione e il deployment
- piattaforme per il monitoraggio del sistema.

2.1. Linguaggi di programmazione

Tecnologia	Versione	Descrizione
Typescript	5.9	Linguaggio di programmazione superset di JavaScript, progettato per aggiungere tipizzazione statica e altre funzionalità. Sviluppo di applicazioni scalabili, miglioramento della qualità del codice, utilizzo con framework come Angular, React e Node.js.
JavaScript	—	Linguaggio di programmazione interpretato, principalmente utilizzato per il lato client nelle applicazioni web. Sviluppo frontend, dinamismo e interattività delle pagine web, utilizzato con framework come React, Vue.js, Angular.
YAML	—	Formato di serializzazione dei dati leggibile dall'uomo, spesso usato per configurazioni. Definizione di file di configurazione, deployment automation, utilizzato con sistemi come Kubernetes, Docker e CI/CD pipelines.
JSON	—	Formato di scambio dati basato su testo, semplice e leggero, ampiamente utilizzato per la comunicazione tra server e client. Scambio di dati tra server e applicazioni web, configurazione di API, utilizzato in contesti come RESTful services e architetture microservizi.

Tabella 1: Linguaggi di programmazione.

2.2. Framework

Tecnologia	Versione	Descrizione
NestJs	11.1.6	Framework per applicazioni Node.js basato su TypeScript, progettato per costruire applicazioni scalabili e modulari. Sviluppo di backend robusti e mantenibili, costruzione di API RESTful, utilizzo con TypeORM, GraphQL, microservizi.

Tabella 2: Frameworks.

2.3. Tecnologie per la gestione dei dati

Tecnologia	Versione	Descrizione
MongoDB	8.0.14	Database NoSQL orientato ai documenti, progettato per gestire grandi volumi di dati non strutturati o semi-strutturati. Archiviazione flessibile di dati JSON-like, scalabilità orizzontale, ideale per applicazioni web moderne, sistemi distribuiti, e gestione di dati in tempo reale.

Tabella 3: Tecnologie per la gestione dei dati.

2.4. Tecnologie per la comunicazione e per la messaggistica

Tecnologia	Versione	Descrizione
NATS	2.11.8	Sistemi di messaggistica leggeri e ad alte prestazioni per la comunicazione asincrona tra microservizi. Gestione di eventi, comunicazione in tempo reale, utilizzato in architetture a microservizi, IoT, e applicazioni distribuite.

Tabella 4: Tecnologie per la comunicazione e per la messaggistica.

2.5. Tecnologie per la virtualizzazione

Tecnologia	Versione	Descrizione
Docker	28.3.2	Piattaforma per la creazione, distribuzione e gestione di container, che permette di isolare applicazioni e dipendenze. Virtualizzazione leggera, gestione di ambienti di sviluppo e produzione, automazione del deployment, utilizzato in DevOps e CI/CD pipelines.

Tabella 5: Tecnologie per la virtualizzazione.

2.6. Tecnologie per il monitoraggio dei microservizi

Tecnologia	Versione	Descrizione
Prometheus	3.6.0	Sistema di monitoraggio e allerta open-source, progettato per raccogliere e archiviare metriche in tempo reale. Utilizza un modello di dati basato su serie temporali e offre potenti funzionalità di query per analizzare le metriche raccolte.
Grafana	12.3.0	Piattaforma open-source per la visualizzazione e l'analisi di dati, spesso utilizzata in combinazione con sistemi di monitoraggio come Prometheus. Consente di creare dashboard interattive e personalizzate per visualizzare metriche, log e altri dati in tempo reale.

Tabella 6: Tecnologie per il monitoraggio dei microservizi.

2.7. Tecnologie per il Testing

Tecnologia	Versione	Descrizione
Jest	30.0	Framework di testing JavaScript sviluppato da Meta, progettato per testare codice JavaScript e TypeScript. Offre funzionalità di test unitari, test di integrazione, mocking e snapshot testing. Utilizzato per garantire affidabilità e qualità del codice nelle applicazioni Node.js e frontend.
Postman	10.16.2	Strumento per il testing e lo sviluppo di API, che consente di creare, testare e documentare API in modo semplice ed efficiente. Offre funzionalità per inviare richieste HTTP, gestire collezioni di API, automatizzare test e monitorare le prestazioni delle API. Utilizzato da sviluppatori e team di sviluppo per garantire la qualità e l'affidabilità delle API. Utilizzato anche per test di accettazione.

Tabella 7: Tecnologie per l'analisi dinamica.

3. Architettura

3.1. Architettura Logica

Il sistema è stato progettato secondo il modello architetturale esagonale (anche conosciuto come Ports and Adapters), che favorisce una netta separazione tra la logica di business centrale e le interazioni con servizi esterni, fonti di dati e interfacce utente.

Questo approccio assicura modularità, testabilità e indipendenza da tecnologie specifiche.

I tre principi fondamentali su cui si basa L'architettura esagonale sono:

- Indipendenza della logica di business: Il cuore dell'applicazione è separato dai dettagli tecnici esterni.

- Porte e adattatori: Le porte definiscono le interfacce che permettono la comunicazione tra il nucleo centrale e l'esterno, mentre gli adattatori implementano queste interfacce per tecnologie specifiche.
- Sostituibilità delle dipendenze: È possibile sostituire facilmente componenti come database, framework o altre dipendenze senza influenzare la logica di business centrale.

E di conseguenza l'architettura si articola in tre componenti principali:

- Core (Dominio e Logica di Business): Il nucleo dell'applicazione che contiene le regole fondamentali e la logica di business, completamente disaccoppiato dalle tecnologie esterne.
- Porte (Ports): Definiscono i punti di accesso e uscita, regolando come il sistema interagisce con l'esterno.
- Adattatori (Adapters): Si occupano di implementare le porte, integrando il sistema con database, servizi esterni e interfacce utente.

Nel contesto di questo modello, le Inbound Ports consentono a componenti esterni di interagire con la logica centrale, mentre le Outbound Ports gestiscono la comunicazione verso i servizi esterni, mantenendo l'astrazione necessaria per preservare l'indipendenza della logica di business. I Services implementano le inbound ports, focalizzandosi esclusivamente sulla logica di dominio senza dipendenze esterne specifiche. Gli Adapters si suddividono in due categorie:

- Input Adapters (o Controllers): Ricevono richieste esterne e le traducono in operazioni comprensibili per il nucleo, invocando le porte in ingresso.
- Output Adapters: Gestiscono la comunicazione con il mondo esterno, traducendo le risposte del nucleo in formati comprensibili dai servizi esterni.

Motivazioni della Scelta Architetturale

La decisione di adottare un'architettura esagonale a microservizi è stata il risultato di un'analisi approfondita dei requisiti del sistema. Le principali ragioni che hanno portato a questa scelta includono:

- Scalabilità e manutenibilità: L'adozione di un'architettura a microservizi consente di gestire in modo più efficace la crescita del sistema, sia in termini di utenti che di funzionalità. Ogni microservizio può evolversi indipendentemente, migliorando la manutenibilità complessiva del sistema e riducendo il rischio di impatti sulle altre componenti.
- Flessibilità tecnologica: L'architettura esagonale a microservizi permette l'adozione di tecnologie diverse per ciascun microservizio, garantendo maggiore libertà nelle scelte tecnologiche e consentendo l'uso di soluzioni più adatte per ogni singolo contesto.
- Indipendenza dei team di sviluppo: Ogni microservizio può essere gestito da team separati, che lavorano in parallelo senza interferire con il lavoro degli altri. Questo migliora la produttività e la velocità di sviluppo, con il vantaggio di poter implementare e distribuire nuove funzionalità senza influenzare l'intero sistema.
- Alta disponibilità e resilienza: L'architettura a microservizi consente di distribuire il carico di lavoro e isolare i guasti. In caso di malfunzionamento di un microservizio, l'impatto sul sistema globale è minimo, migliorando la resilienza complessiva.
- Possibilità di evoluzione continua: Adottando un'architettura esagonale con microservizi, è possibile evolvere gradualmente il sistema, aggiungendo nuove funzionalità o migliorando

quelle esistenti senza dover riscrivere completamente l'applicazione. Inoltre, è possibile scalare i microservizi in modo indipendente in base al carico di lavoro.

- Adattamento alle esigenze future: L'architettura a microservizi offre la possibilità di integrare facilmente nuove tecnologie, strumenti o piattaforme, senza compromettere la stabilità dell'intero sistema. Ciò consente al sistema di evolvere in modo agile e rispondere rapidamente alle esigenze del business.

In conclusione, l'architettura esagonale rappresenta una scelta ideale per garantire modularità, sostenibilità a lungo termine e un'elevata capacità di adattamento alle esigenze future del sistema.

3.2. Pattern Architetturali

3.2.1. Sistema a microservizi

3.2.1.1. Descrizione del pattern Microservizi

Il pattern Microservizi è un'architettura che suddivide un'applicazione complessa in una serie di servizi indipendenti, ciascuno responsabile di una specifica funzionalità di business. Ogni microservizio è autonomo, può essere sviluppato, distribuito e scalato in modo indipendente dagli altri, comunica tramite API leggere (tipicamente HTTP/REST o messaggistica asincrona come con NATS) e possiede il proprio modello di dati e logica di persistenza. Questo approccio favorisce la modularità, la resilienza e la possibilità di adottare tecnologie differenti per ciascun servizio, facilitando l'evoluzione e la manutenzione del sistema.

3.2.1.2. Motivazioni dell'utilizzo del pattern Microservizi

L'adozione del pattern Microservizi è stata motivata sia dalla richiesta esplicita dell'azienda proponente, sia dall'analisi delle esigenze di un sistema a magazzini distribuiti. Questo approccio si è rivelato particolarmente adatto per diversi motivi:

- **Scalabilità**: ogni magazzino può essere gestito come un servizio indipendente, facilitando l'espansione del sistema e la gestione di carichi variabili.
- **Resilienza**: la suddivisione in servizi autonomi riduce l'impatto di eventuali guasti, migliorando la disponibilità complessiva.
- **Manutenibilità**: i microservizi consentono aggiornamenti e modifiche localizzate senza influenzare l'intero sistema.
- **Flessibilità tecnologica**: ogni servizio può essere sviluppato e aggiornato con tecnologie e linguaggi diversi, scegliendo la soluzione più adatta al contesto specifico.
- **Indipendenza dei team**: lo sviluppo parallelo di più microservizi permette una maggiore produttività e una migliore organizzazione del lavoro.

In sintesi, il pattern Microservizi garantisce modularità, adattabilità e robustezza, risultando la scelta ideale per un sistema distribuito come quello richiesto dal progetto.

3.2.2. Architettura esagonale

3.2.2.1. Descrizione dell'architettura esagonale

L'architettura esagonale, nota anche come Ports and Adapters, è un modello architetturale che promuove la separazione tra la logica di business centrale e le interazioni con servizi esterni, fonti di dati e interfacce utente. Questo approccio consente di creare sistemi modulari, testabili e indipendenti da tecnologie specifiche.

3.2.2.2. Motivazioni dell'utilizzo dell'architettura esagonale

L'architettura esagonale è stata adottata per i microservizi perché garantisce una netta separazione tra la logica di business e le dipendenze esterne (database, sistemi di messaggistica, API, ecc.), favorendo modularità, testabilità e indipendenza tecnologica. In un contesto a microservizi, questa separazione permette di sviluppare, testare e distribuire ogni servizio in modo autonomo, semplificando l'integrazione con tecnologie diverse e facilitando la sostituzione o l'evoluzione dei componenti esterni senza impattare la logica centrale. Inoltre, l'approccio Ports & Adapters consente di mantenere i microservizi facilmente estendibili e manutenibili, riducendo il rischio di accoppiamento tra le parti e migliorando la resilienza complessiva del sistema.

3.2.3. Saga Pattern

3.2.3.1. Descrizione del Saga Pattern

È un pattern architetturale usato per gestire le transazioni distribuite nei sistemi a microservizi. Una Saga suddivide una transazione complessa in una serie di transazioni locali, ognuna gestita da un servizio. Se una di queste fallisce, vengono eseguite delle operazioni di compensazione per annullare le modifiche precedenti e mantenere la consistenza dei dati.

3.2.3.2. Motivazioni dell'utilizzo del Saga Pattern

Nel progetto dei magazzini distribuiti molte operazioni coinvolgono più servizi (ad esempio spostare scorte da un magazzino a un altro, o aggiornare disponibilità e ordini). Non potendo fare affidamento su una transazione globale, il Saga pattern garantisce che il sistema resti consistente tramite compensazioni. Questo rende l'architettura più affidabile e resiliente ai fallimenti parziali, evitando blocchi o incoerenze nei dati. Nel nostro sistema abbiamo scelto di seguire un saga pattern orchestrato dove un servizio gestisce il processo delle transizioni della saga.

3.2.4. CQRS (Command Query Responsibility Segregation)

3.2.4.1. Descrizione del pattern CQRS

Il pattern CQRS separa le operazioni di scrittura (Command) da quelle di lettura (Query), utilizzando modelli e percorsi diversi per ciascun tipo di operazione. I comandi modificano lo stato del sistema, mentre le query si occupano esclusivamente di restituire dati, senza effetti collaterali. Questo approccio permette di ottimizzare e scalare in modo indipendente le operazioni di lettura e scrittura.

3.2.4.2. Motivazioni dell'utilizzo del pattern CQRS

Nel progetto, la gestione dei dati di magazzino richiede alta scalabilità: le scritture (aggiornamenti di scorte, inserimento ordini) sono meno frequenti ma critiche, mentre le letture (consultazione scorte, statistiche, stato ordini) sono molto più numerose. L'adozione di CQRS consente di ottimizzare le performance di entrambe le parti, facilitando l'integrazione di sistemi distribuiti e l'uso di cache (es. Redis) per le query, senza impattare la coerenza e l'affidabilità delle operazioni di scrittura.

3.2.5. Event Sourcing

3.2.5.1. Descrizione del pattern Event Sourcing

È un pattern in cui lo stato del sistema non viene salvato direttamente come uno snapshot finale, ma come una sequenza di eventi immutabili che rappresentano ogni cambiamento avvenuto nel

sistema. Lo stato corrente può sempre essere ricostruito rigiocando in ordine tutti gli eventi registrati.

3.2.5.2. Motivazioni dell'utilizzo del pattern Event Sourcing

Nei magazzini distribuiti è fondamentale tenere traccia di ogni cambiamento (ad esempio: arrivo merce, spostamento scorte, evasione di un ordine). L'adozione di Event Sourcing consente di mantenere uno storico completo e auditabile di tutte le operazioni, risultando prezioso per attività di debug, analisi, resilienza e per la ricostruzione dei dati in caso di inconsistenze o errori.

3.3. Design patterns

3.3.1. Domain-Driven Design (DDD)

3.3.1.1. Descrizione del Domain-Driven Design (DDD)

È un approccio alla progettazione software che mette al centro il dominio applicativo (in questo caso la gestione dei magazzini) e il linguaggio ubiquitario, cioè un linguaggio condiviso tra sviluppatori e stakeholder. Divide il sistema in bounded context, ognuno con il proprio modello coerente.

3.3.1.2. Motivazioni dell'utilizzo del Domain-Driven Design (DDD)

Nel progetto dei magazzini distribuiti il dominio è complesso (scorte, sincronizzazione, logistica, ridondanza dei dati). Usare DDD permette di mantenere un modello chiaro e condiviso, riducendo ambiguità e rendendo più facile la comunicazione con l'azienda partner. Inoltre, separando i bounded context (es. gestione ordini, inventario, sincronizzazione) si ottiene un'architettura modulare e più manutenibile.

3.3.2. Dependency Injection (DI)

3.3.2.1. Descrizione del pattern Dependency Injection (DI)

È un pattern che permette di gestire le dipendenze tra oggetti dall'esterno, invece che crearle direttamente nel codice. Le classi dichiarano le loro dipendenze e un framework o un contenitore DI le fornisce.

3.3.2.2. Motivazioni dell'utilizzo del pattern Dependency Injection (DI)

In un sistema a microservizi e architettura modulare, la DI semplifica i test (mock delle dipendenze), migliora la manutenibilità e rende i servizi meno accoppiati. Questo è particolarmente utile per il progetto, dove i vari moduli (es. sincronizzazione, gestione ordini, API) devono poter evolvere indipendentemente. Inoltre NestJs supporta nativamente la DI, facilitando l'implementazione di questo pattern.

3.4. Microservizi sviluppati

Ogni microservizio è stato progettato per assolvere a un compito ben definito, garantendo un contributo essenziale all'interno del sistema complessivo del progetto e favorendo la collaborazione tra i vari moduli. L'architettura dei **Microservizi** segue un approccio a casi d'uso e servizi di dominio, con separazione tra logica applicativa, logica di dominio e interfacce di comunicazione esterne.
I microservizi sviluppati sono:

- **Inventory**
- **State**
- **Cloud State**

- **Orders**
- **Inventory Aggregate**
- **Orders Aggregate**
- **Central System (Sistema Centralizzato)**
- **Authentication**
- **Routing**

Ogni microservizio possiede il proprio file

- **service** : Contiene la logica applicativa principale, orchestrando i casi d'uso e coordinando le entità di dominio.
- **DTO** : Strutture leggere dedicate allo scambio di dati tra microservizi o verso l'esterno, senza esporre direttamente le entità di dominio.
- **DataMapper** : Componente responsabile della conversione bidirezionale tra entità di dominio e DTO. Garantisce separazione tra logica interna e interfacce esterne.
- **Adapter** : Moduli che implementano le interfacce per la comunicazione esterna (eventi, API, messaggistica). Seguono il pattern Ports & Adapters.
- **Controller** : Punto di ingresso del microservizio per la gestione delle richieste provenienti dall'esterno (es. REST controller, listener di eventi).
- **Repository** : Strato di accesso ai dati per la gestione della persistenza delle entità di dominio.
  - Nota: il *Sistema Centralizzato* non possiede la repository perché non ha persistenza locale ma agisce come orchestratore.
- **Domini** : Le entità e i value object che rappresentano il cuore del microservizio. Sono indipendenti dalla tecnologia e descrivono il linguaggio del dominio

3.4.1. Microservizio Inventario (Inventory Service)

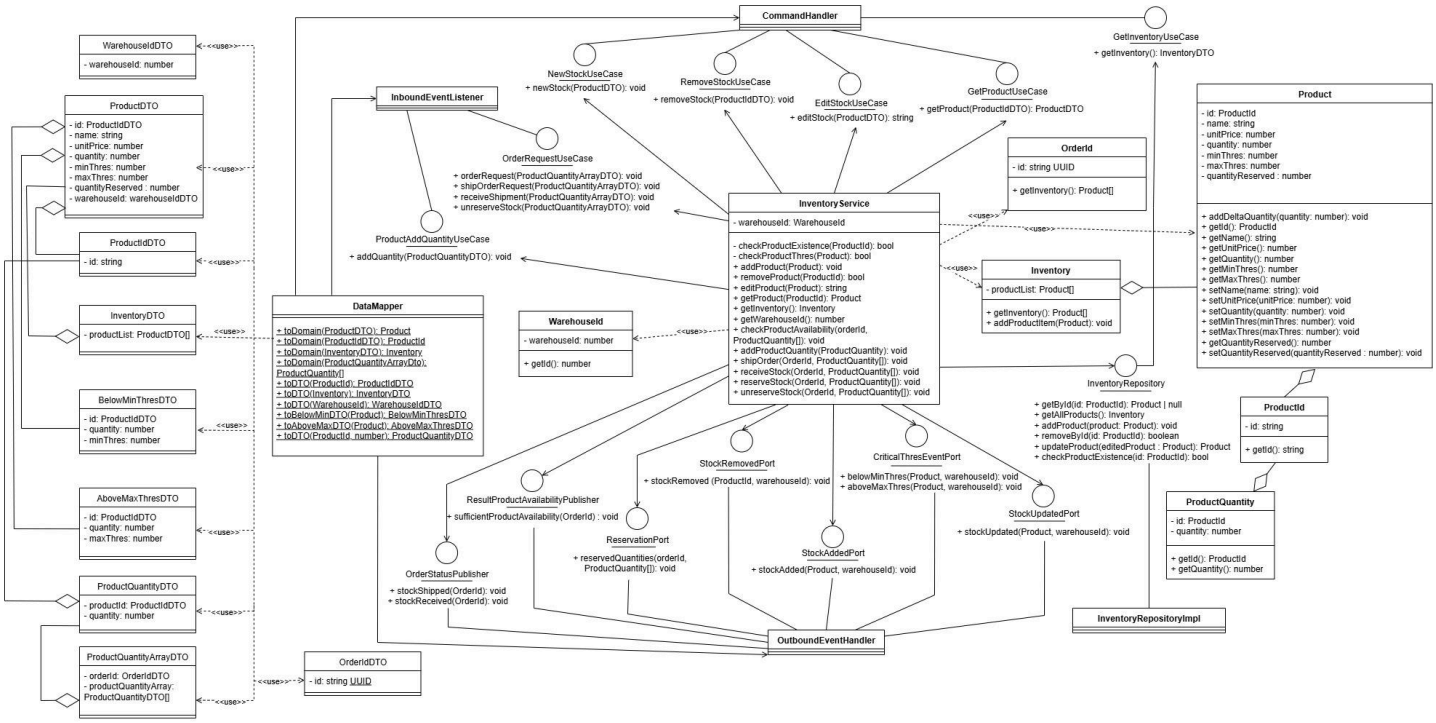

Figura 1: Schema UML - Microservizio Inventario

3.4.1.1. Descrizione del microservizio

Il **Microservizio Inventario** rappresenta il componente responsabile della gestione delle scorte all'interno di un singolo magazzino. Il suo compito principale è quello di mantenere la coerenza dello stato dei prodotti, verificare la disponibilità rispetto alle soglie minime e massime configurate e coordinare le operazioni di movimentazione delle quantità.

3.4.1.1.1. Funzionalità principali

- **Gestione dello stock prodotti**: inserimento, rimozione, aggiornamento e consultazione delle quantità.
- **Verifica disponibilità**: controllo delle soglie di giacenza (minime e massime) e identificazione di eventuali criticità.
- **Gestione richieste ordini**: elaborazione delle richieste provenienti dai magazzini o dal sistema centralizzato per soddisfare ordini cliente.
- **Pubblicazione eventi**: emissione di notifiche verso gli altri microservizi quando avvengono variazioni di stato significative.
- **Interoperabilità**: interazione con microservizi esterni quali:
  - *Sistema Centralizzato*, che coordina gli eventi critici e la logica di alto livello.
  - *Microservizio Ordini*, per validare la disponibilità dei prodotti richiesti.
  - *Inventario Aggregato*, per fornire una vista complessiva delle giacenze multi-magazzino.

3.4.1.2. ProductId

1. Rappresenta l'identificatore univoco del prodotto,
2. Incapsula il campo *id: string*,
3. Espone il metodo *getId()*,
4. È stato isolato per facilitare il confronto tra entità e mantenere l'identità coerente anche in fase di serializzazione/deserializzazione *(es. tramite DTO)*, ad esempio per permettere il corretto funzionamento di *removeProduct*.

Descrizione degli attributi della struttura:
- **id**: string
  È il codice univoco del prodotto.

Può invocare le seguenti funzioni:
- **getId()**: string
  Metodo pubblico per ottenere il codice univoco del prodotto.

3.4.1.3. Product

1. Contiene tutti gli attributi modificabili del prodotto: *name, unitPrice, quantity, minThres, maxThres*,
2. I campi *name* e *unitPrice* sono modificabili solo dal Supervisore Globale in quanto dati sensibili e che verranno riflessi in tutit i magazzini in caso di modifica,
3. I campi *quantity*, *minThres*, *maxThres* sono modificabili anche dai Supervisori Locali: tali modifiche non verranno riflesse negli altri magazzini ma riguarderanno solo il magazzino di pertinenza.

Descrizione degli attributi della struttura:
- **id**: ProductId
  Rappresenta l'Id del prodotto.
- **name**: string
  Rappresenta il nome del prodotto.
- **unitPrice**: number
  Rappresenta il prezzo unitario del prodotto.
- **quantity**: number
  È la quantità attualmente disponibile del prodotto.
- **minThres**: number
  Soglia minima di sicurezza relativa alla quantità del prodotto.
- **maxThres**: number
  Soglia massima di sicurezza relativa alla quantità del prodotto.
- **quantityReserved**: number
  È la quantità riservata di quel prodotto. Può essere NULL.

Può invocare le seguenti funzioni:
- **addDeltaQuantity(quantity: number)**: void
  Metodo per sommare o sottrarre una quantità positiva ad un prodotto.
- **getId()**: ProductId;
  Restituisce il codice univoco del prodotto.
- **getName()**: string;
  Restituisce il nome del prodotto.
- **getUnitPrice()**: number;
  Restituisce il prezzo unitario del prodotto.
- **getQuantity()**: number;
  Restituisce la quantità disponibile del prodotto.
- **getMinThres()**: number;
  Restituisce la soglia minima di sicurezza relativa alla quantità del prodotto.
- **getMaxThres()**: number;
  Restituisce la soglia massima di sicurezza relativa alla quantità del prodotto.
- **setName(name: string)**: void;
  Restituisce il nome del prodotto.

- **setUnitPrice(unitPrice: number)**: void;
  Metodo che permette di modificare il prezzo unitario.
- **setQuantity(quantity: number)**: void;
  Metodo per modificare la quantità disponibile del prodotto.
- **setMinThres(minThres: number)**: void;
  Metodo per modificare la soglia minima di sicurezza relativa alla quantità del prodotto.
- **setMaxThres(maxThres: number)**: void;
  Metodo per modificare la soglia massima di sicurezza relativa alla quantità del prodotto.
- **getQuantityReserved()**: number
  Restituisce la quantità riservata del prodotto.
- **setQuantityReserved(quantityReserved : number)**: void Metodo per modificare la quantità riservata del prodotto.

Questa separazione segue i principi della **Domain-Driven Design** (DDD): i DTO vengono convertiti in entità complete (`Product`) già nei livelli più alti (es. nel controller), evitando che il livello **Application Service** debba conoscere i dettagli della rappresentazione dati in entrata/uscita. Si garantisce così la **separazione dei livelli** e una migliore **manutenibilità** del codice.

3.4.1.4. ProductQuantity

1. Rappresenta un oggetto di dominio che associa l'id di un prodotto ad una quantità numerica.
2. Non ha un uso specifico ma è utilizzabile in più contesti.

Descrizione degli attributi della struttura:
- **id**: ProductId Rappresenta l'id del prodotto.
- **quantity**: number Rappresenta la quantità numerica associata al prodotto.

Può invocare le seguenti funzioni:
- **getId()**: ProductId;
  Restituisce il codice univoco del prodotto.
- **getQuantity()**: number
  Restituisce la quantità associata al prodotto.

3.4.1.5. Inventory

1. Rappresenta una lista di prodotti.
2. Usato principalmente per restituire l'intero inventario di un magazzino.

Descrizione degli attributi della struttura:
- **productList**: Product[] Rappresenta la lista di prodotti.

Può invocare le seguenti funzioni:
- **getInventory()**: ProductId[];
  Restituisce la lista di prodotti.
- **addProductItem(Product)**:
  Aggiunge un prodotto nella lista di prodotti.

3.4.1.6. OrderId

1. Rappresenta l'id di un ordine.
2. È incluso anche nel servizio di inventario perché è necessario per mantenere l'assocazione tra un ordine e gli elementi di un ordine nella comunicazione asincrona con il microservizio di Ordini.
3. Il carattere iniziale rappresenta il tipo di ordine. Questo è molto utile per identificare il tipo di ordine per logiche di controllo.

Descrizione degli attributi della struttura:
- **id**: string

Rappresenta l'id di un ordine; il primo carattere rappresenta il tipo di ordine, i rimanenti sono un UUID

Può invocare le seguenti funzioni:
- getId(): string

Restituisce l'id dell'ordine
- getOrderType(): string

Restituisce il tipo di ordine

3.4.1.7. InventoryService

1. Rappresenta il servizio applicativo responsabile della logica di business relativa alla gestione dell'inventario di un magazzino.
2. Espone operazioni per l'aggiunta, rimozione, modifica e consultazione dei prodotti, oltre a controlli di soglia e disponibilità.
3. Ogni modifica ai prodotti genera un evento che viene propagato al sistema.

Descrizione degli attributi della struttura:
- **warehouseId**: WarehouseId
  Identificativo univoco del magazzino a cui appartiene l'inventario gestito dal servizio.

Può invocare le seguenti funzioni:
- **checkProductExistence(productId: ProductId)**: bool
  Verifica se un prodotto è presente nell'inventario del magazzino.
- **checkProductThres(product: Product)**: bool
  Controlla se la quantità del prodotto rispetta le soglie minime e massime definite.
- **addProduct(product: Product)**: void
  Aggiunge un nuovo prodotto all'inventario e pubblica un evento di aggiunta.
- **removeProduct(productId: ProductId)**: bool
  Rimuove un prodotto dall'inventario e pubblica un evento di rimozione. Restituisce true se l'operazione è andata a buon fine.
- **editProduct(product: Product)**: void
  Modifica i dati di un prodotto già presente e pubblica un evento di modifica.
- **getProduct(productId: ProductId)**: Product
  Restituisce il prodotto corrispondente all'Id specificato.
- **getInventory()**: Inventory
  Restituisce la lista completa dei prodotti presenti nell'inventario.
- **getWarehouseId()**: number
  Restituisce l'identificativo del magazzino.
- **checkProductAvailability(productQuantities: ProductQuantity[])**: bool
  Verifica la disponibilità di uno o più prodotti nelle quantità richieste.
- **addProductQuantity(ProductQuantity)**: void Aggiunge una quantità numerica alla quantità disponibile un prodotto.
- **shipOrder(OrderId, ProductQuantity[])**: void Spedisce l'ordine sottraendo le quantità richieste da quelle riservate e pubblica un evento di avvenuta spedizione.
- **receiveStock(OrderId, ProductQuantity[])**: void Processa il ricevimento di merce da parte di un ordine e pubblica un evento di avvenuta ricezione.

- **reserveStock(OrderId, ProductQuantity**[]): void Riserva la merce richiesta da un ordine e pubblica un evento con le quantità riservate.

3.4.1.8. WarehouseId

1. È stato separato da `InventoryService` in quanto l'identificativo del magazzino è necessario principalmente in fase di sincronizzazione sul cloud durante operazioni di modifica, aggiunta o rimozione di prodotti.
2. Non è quindi necessario che il DTO di `InventoryService` contenga sempre l'id del magazzino.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  Identificativo univoco del magazzino.

Può invocare le seguenti funzioni:

- **getId()**: number
  Restituisce l'identificativo del magazzino.

3.4.1.9. WarehouseIdDTO

Descrizione degli attributi della struttura:

- **warehouseId**: number
  Identificativo del magazzino, rappresentato come DTO.

3.4.1.10. OrderIdDTO

Descrizione degli attributi della struttura:

- **id**: string
  Identificativo dell'ordine, rappresentato come DTO.

3.4.1.11. ProductIdDTO

Descrizione degli attributi della struttura:

- **id**: string
  Identificativo del prodotto, rappresentato come DTO.

3.4.1.12. ProductDTO

Descrizione degli attributi della struttura:

- **id**: string
  Identificativo del prodotto, rappresentato come DTO.
- **name**: string
  Nome del prodotto, rappresentato come DTO.
- **unitPrice**: number
  Prezzo unitario del prodotto, rappresentato come DTO.
- **quantity**: number
  Quantità attualmente disponibile del prodotto, rappresentata come DTO.
- **minThres**: number
  Soglia minima di sicurezza relativa alla quantità del prodotto, rappresentata come DTO.
- **maxThres**: number
  Soglia massima di sicurezza relativa alla quantità del prodotto, rappresentata come DTO.
- **quantityReserved**: number Quantità riservata nel magazzino di un prodotto.
- **warehouseId**: warehouseIdDTO Identificativo del magazzino.

3.4.1.13. InventoryDTO

Descrizione degli attributi della struttura:

- **productList**: ProductDTO[]
  Elenco dei prodotti presenti nell'inventario, rappresentato come DTO.

3.4.1.14. BelowMinThresDTO

Descrizione degli attributi della struttura:

- **id**: string
  Identificativo del prodotto, rappresentato come DTO.
- **quantity**: number
  Quantità attualmente disponibile del prodotto, rappresentata come DTO.
- **minThres**: number
  Soglia minima di sicurezza del prodotto, rappresentata come DTO.

3.4.1.15. AboveMaxThresDTO

Descrizione degli attributi della struttura:

- **id**: string
  Identificativo del prodotto, rappresentato come DTO.
- **quantity**: number
  Quantità attualmente disponibile del prodotto, rappresentata come DTO.
- **maxThres**: number
  Soglia massima di sicurezza del prodotto, rappresentata come DTO.

3.4.1.16. ProductQuantityDTO

1. Rappresenta una coppia *(prodotto, quantità)* utilizzata nei casi d'uso e nella comunicazione tra livelli applicativi.

Descrizione degli attributi della struttura:

- **productId**: string
  Identificativo del prodotto, rappresentato come DTO.
- **quantity**: number
  Quantità richiesta o disponibile del prodotto, rappresentata come DTO.

3.4.1.17. ProductQuantityArrayDTO

1. Associa un ordine ad un array di coppie *(prodotto, quantità)*, utile per operazioni batch (es. richieste d'ordine).

Descrizione degli attributi della struttura:

- **orderId**: OrderIdDTO
  Rappresenta l'id dell'ordine.
- **productQuantityArray**: ProductQuantityDTO[]
  Elenco delle quantità dei prodotti, rappresentato come DTO.

3.4.1.18. DataMapper

1. Classe di utilità che si occupa della conversione tra DTO e oggetti di dominio.
2. Garantisce l'isolamento del dominio dai dettagli di rappresentazione dati (DTO).

Metodi statici:

- **toDomain(productDTO: ProductDTO)**: Product
  Converte un DTO ProductDTO nell'entità di dominio *Product*.
- **toDomain(productIdDTO: ProductIdDTO)**: ProductId
  Converte un DTO ProductIdDTO nell'oggetto di dominio *ProductId*.

- **toDomain(inventoryDTO: InventoryDTO)**: Inventory
  Converte un DTO InventoryDTO nell'oggetto di dominio *Inventory*.
- **toDTO(product: Product)**: ProductDTO
  Converte un'entità di dominio Product nel corrispondente *ProductDTO*.
- **toDTO(productId: ProductId)**: ProductIdDTO
  Converte un oggetto di dominio ProductId nel corrispondente *ProductIdDTO*.
- **toDTO(inventory: Inventory)**: InventoryDTO
  Converte un oggetto di dominio Inventory nel corrispondente *InventoryDTO*.
- **toDTO(warehouseId: WarehouseId)**: WarehouseIdDTO
  Converte un oggetto di dominio WarehouseId nel corrispondente *WarehouseIdDTO*.
- **toBelowMinDTO(product: Product)**: BelowMinThresDTO
  Converte un'entità Product in BelowMinThresDTO, filtrando i campi non rilevanti *(es. name, unitPrice)*.
- **toAboveMaxDTO(product: Product)**: AboveMaxThresDTO
  Converte un'entità Product in AboveMaxThresDTO, filtrando i campi non rilevanti *(es. name, unitPrice)*.
- **toDTO(productId: ProductId, quantity: number)**: QuantityRequestedDTO
  Converte una coppia (ProductId, quantità) in un *QuantityRequestedDTO*.

3.4.1.19. ProductAddQuantityUseCase

1. Definisce il caso d'uso relativo all'aggiunta di una quantità a un prodotto già presente nell'inventario.

Metodi:

- **addQuantity(productQuantityDTO: ProductQuantityDTO)**: void
  Aggiunge la quantità specificata al prodotto indicato, previa conversione del DTO in *ProductQuantity[]*.

3.4.1.20. OrderRequestUseCase

1. Definisce il caso d'uso relativo alla gestione delle richieste d'ordine *(es. richiesta multiprodotto)*.

Metodi:

- **orderRequest(productQuantityArrayDTO: ProductQuantityArrayDTO)**: void
  Gestisce la richiesta iniziale di ordine a partire da un array di coppie *(prodotto, quantità)*.
- **shipOrderRequest(ProductQuantityArrayDTO)**: void Processa l'evento di spedire della merce per un ordine.
- **receiveShipment(ProductQuantityArrayDTO)**: void Processa l'evento di ricevere della merce per via di un ordine.

3.4.1.21. InboundEventHandler

1. Responsabile della gestione degli eventi in ingresso *(ad esempio, eventi provenienti da altri servizi o dal cloud)*.
2. Coordina l'invocazione dei casi d'uso applicativi in risposta a tali eventi.

3.4.1.22. NewStockUseCase

1. Definisce il caso d'uso relativo all'aggiunta di un nuovo prodotto all'interno dell'inventario.

Metodi:

- **newStock(productDTO: ProductDTO)**: void
  Aggiunge un nuovo prodotto all'inventario, previa conversione del DTO in *Product*.

3.4.1.23. RemoveStockUseCase

1. Definisce il caso d'uso relativo alla rimozione di un prodotto dall'inventario.

Metodi:

- **removeStock(productIdDTO: ProductIdDTO)**: void
  Rimuove un prodotto dall'inventario, previa conversione del DTO in *ProductId*.

3.4.1.24. EditStockUseCase

1. Definisce il caso d'uso relativo alla modifica dei dati di un prodotto nell'inventario.

Metodi:

- **editStock(productDTO: ProductDTO)**: void
  Modifica uno o più campi di un prodotto esistente, previa conversione del DTO in *Product*.

3.4.1.25. GetProductUseCase

1. Definisce il caso d'uso relativo all'ottenimento di un prodotto specifico dall'inventario.

Metodi:

- **getProduct(productIdDTO: ProductIdDTO)**: void
  Restituisce un prodotto a partire dal suo identificativo, previa conversione del DTO in *ProductId*.

3.4.1.26. GetInventoryUseCase

1. Definisce il caso d'uso relativo all'ottenimento della lista completa dei prodotti presenti nell'inventario.

Metodi:

- **getInventory()**: void
  Restituisce l'elenco dei prodotti dell'inventario.

Queste interfacce definiscono i **casi d'uso dell'applicazione**, e sono tipicamente implementate da un Application Service (es. `InventoryService`, nel caso del PoC, `inventoryHandler.service`), che coordina la logica e interagisce col dominio.

3.4.1.27. CommandHandler

1. Responsabile della gestione degli eventi in ingresso.
2. Implementa i casi d'uso: NewStockUseCase, EditStockUseCase, RemoveStockUseCase, GetProductUseCase e GetInventoryUseCase.
3. Riceve i DTO dall'esterno, li converte negli oggetti di dominio corrispondenti e li inoltra tramite la relativa chiamata all'InventoryService.
4. In questo modo funge da adattatore di ingresso (Input Port), mantenendo separati i dettagli di trasporto dei dati dalla logica di business.

3.4.1.28. ResultProductAvailabilityPublisher

1. Definisce la porta di uscita per la pubblicazione di eventi dell'esito di riservamento prodotti.

Metodi:

- **sufficientProductAvailability(OrderId)**: void Pubblica un evento per comunicare l'avvenuto riservamento di tutte le quantità richieste per tutti i prodotti di un ordine.

3.4.1.29. OrderStatusEventPublisher

1. Definisce la porta di uscita per la pubblicazione di eventi relativi agli aggiornamenti di stato di un ordine (*e.g.* spedizione)

- **stockShipped(OrderId)**: void

Pubblica l'evento di avvenuta spedizione della merce relativa a un ordine.

- **stockReceived(OrderId)**: void

Pubblica l'evento di avvenuta ricezione di merce da parte di un ordine in entrata.

3.4.1.30. ReservationPort

1. Definisce la porta di uscita per la pubblicazione di eventi di riservamento prodotti.

Metodi:

- **reservedQuantities(orderId: OrderId, pq: ProductQuantity[])**: void
  Pubblica un evento che notifica le quantità di prodotti riservate per un ordine.

3.4.1.31. CriticalThresEventPort

1. Definisce la porta di uscita per la pubblicazione di eventi relativi al superamento delle soglie critiche di inventario.

Metodi:

- **belowMinThres(product: Product)**: void
  Pubblica un evento che notifica il superamento della soglia minima di sicurezza per un prodotto.
- **aboveMaxThres(product: Product)**: void
  Pubblica un evento che notifica il superamento della soglia massima di sicurezza per un prodotto.

3.4.1.32. StockAddedPort

1. Definisce la porta di uscita per la pubblicazione di eventi di aggiunta di merce nell'inventario.

Metodi:

- **stockAdded(product: Product, warehouseId: WarehouseId)**: void
  Pubblica un evento che notifica l'aggiunta di un prodotto all'interno di un magazzino.

3.4.1.33. StockRemovedPort

1. Definisce la porta di uscita per la pubblicazione di eventi di rimozione di merce dall'inventario.

Metodi:

- **stockRemoved(productId: ProductId, warehouseId: WarehouseId)**: void
  Pubblica un evento che notifica la rimozione di un prodotto dall'inventario di un magazzino.

3.4.1.34. StockUpdatedPort

1. Definisce la porta di uscita per la pubblicazione di eventi di aggiornamento dei dati di un prodotto.

Metodi:

- **stockUpdated(product: Product, warehouseId: WarehouseId)**: void
  Pubblica un evento che notifica la modifica dei dati di un prodotto presente in un magazzino.

3.4.1.35. GetProductPort

1. Definisce la porta di uscita per la pubblicazione di eventi relativi all'ottenimento di un prodotto specifico.

Metodi:

- **publishProduct(product: Product)**: void
  Pubblica l'evento che notifica la disponibilità dei dati di un prodotto.

3.4.1.36. GetInventoryPort

1. Definisce la porta di uscita per la pubblicazione di eventi relativi all'ottenimento dell'inventario completo.

Metodi:

- **publishInventory(inventory: Inventory, warehouseId: WarehouseId)**: void
  Pubblica l'evento che notifica il contenuto dell'inventario di un magazzino.

3.4.1.37. OutboundEventAdapter

1. Adattatore responsabile della gestione degli eventi in uscita.
2. Implementa le interfacce: *CriticalThresEventPort, StockAddedPort, StockRemovedPort, StockUpdatedPort, GetProductPort e GetInventoryPort.*
3. Converte gli oggetti di dominio nei rispettivi DTO prima della pubblicazione degli eventi.

Questi porti rappresentano **eventi di dominio (Output Port)**, utili per notificare modifiche o stati ad altri moduli del sistema. Tale struttura segue i principi dell'architettura esagonale, garantendo separazione tra:

- valori immutabili e mutabili del dominio,
- entità *(Product)*,
- interfacce di comando ed evento.

Ne deriva una maggiore **testabilità e manutenibilità** del sistema.

3.4.1.38. InventoryRepository

1. Definisce l'interfaccia del repository per la gestione dei prodotti di un inventario.
2. Fornisce metodi per recupero, persistenza e controllo dell'esistenza dei prodotti.

Metodi:

- **getById(id: ProductId)**: Product | null
  Restituisce un prodotto a partire dal suo identificativo, oppure null se non esiste.
- **getAllProducts()**: Inventory
  Restituisce l'inventario completo del magazzino.
- **addProduct(product: Product)**: Product
  Aggiunge un nuovo prodotto all'inventario e restituisce il prodotto inserito.
- **removeById(id: ProductId)**: boolunivoco
  Rimuove un prodotto a partire dal suo identificativo e restituisce true se l'operazione ha avuto successo.
- **updateProduct(editedProduct: Product)**: Product | null
  Aggiorna i dati di un prodotto esistente e restituisce l'oggetto aggiornato, oppure null se non trovato.
- **checkProductExistence(id: ProductId)**: bool
  Verifica l'esistenza di un prodotto nell'inventario del magazzino.

3.4.1.39. InventoryRepositoryImpl

1. Implementazione concreta di InventoryRepository.
2. Si occupa della persistenza dei prodotti (es. database locale o sincronizzazione cloud).
3. Incapsula i dettagli tecnologici, mantenendo il dominio indipendente dalle scelte infrastrutturali.

3.4.2. Microservizio State (Warehouse State Service)

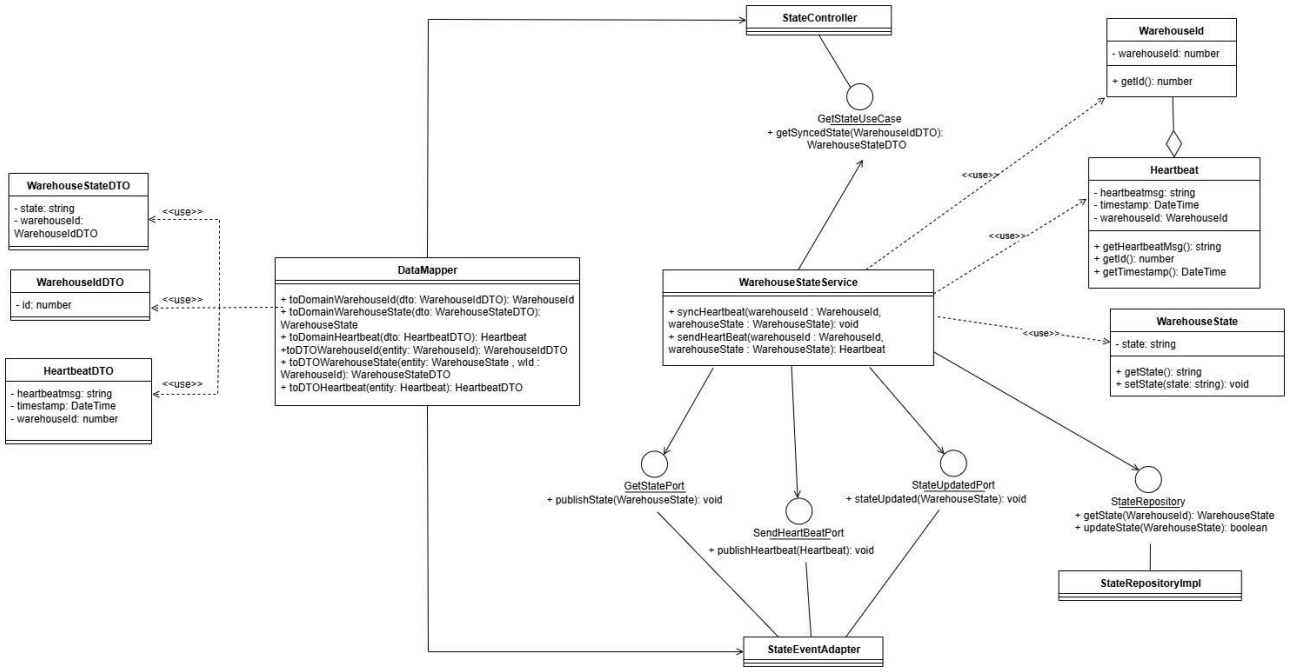

Figura 2: Schema UML - Microservizio Stato locale

3.4.2.1. Descrizione del microservizio

Il **Microservizio Stato** rappresenta il componente responsabile del monitoraggio e della gestione dello stato di un magazzino. Il suo compito principale è quello di rilevare la disponibilità operativa attraverso heartbeat periodici, mantenere aggiornato lo stato corrente e inviare tali informazioni al microservizio **Cloud State**.

3.4.2.1.1. Funzionalità principali

- **Gestione heartbeat**: raccolta, validazione e aggiornamento delle informazioni di stato provenienti dal magazzino.
- **Aggiornamento stato**: mantenimento della coerenza delle informazioni sullo stato operativo del magazzino.
- **Trasmissione a Cloud State**: invio degli heartbeat al Cloud State per l'elaborazione e il coordinamento.
- **Interoperabilità**: interazione con microservizi esterni quali:
  - *Cloud State*, per la gestione e il coordinamento degli stati.

3.4.2.2. WarehouseId

1. Rappresenta l'identificatore univoco del magazzino,
2. Incapsula il campo *warehouseId: number*,
3. Espone il metodo *getId()*,
4. È stato isolato per facilitare il confronto tra entità e mantenere l'identità coerente anche in fase di serializzazione/deserializzazione.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  È l'identificativo numerico del magazzino.

Può invocare le seguenti funzioni:

- **getId()**: number
  Metodo pubblico per ottenere l'id del magazzino.

3.4.2.3. WarehouseState

1. Contiene lo stato corrente del magazzino,
2. Può essere utilizzato per monitorare lo stato di presenza del magazzino.

Descrizione degli attributi della struttura:

- **state**: string
  Rappresenta lo stato del magazzino.

Può invocare le seguenti funzioni:

- **getState()**: string
  Metodo che restituisce lo stato corrente del magazzino.

3.4.2.4. Heartbeat

1. Rappresenta un messaggio periodico di stato inviato dal magazzino,
2. Contiene informazioni temporali e di identificazione del magazzino.

Descrizione degli attributi della struttura:

- **heartbeatMsg**: string
  Contenuto del messaggio di stato del magazzino.
- **timestamp**: DateTime
  Data e ora di emissione del messaggio.
- **warehouseId**: WarehouseId
  Identificativo del magazzino mittente.

Può invocare le seguenti funzioni:

- **getHeartbeatMsg()**: string
  Restituisce il messaggio di stato.
- **getId()**: number
  Restituisce l'id del magazzino.
- **getTimestamp()**: DateTime
  Restituisce la data e l'ora del messaggio.

3.4.2.5. WarehouseStateService

1. Costituisce la logica di business del servizio

Descrizione degli attributi della struttura:

- **heartbeat**: Heartbeat
  Rappresenta lo stato corrente del magazzino.

Può invocare le seguenti funzioni:

- **syncHeartbeat(heartbeat: Heartbeat)**: void
  Sincronizza lo stato del magazzino con il messaggio ricevuto.
- **sendHeartBeat(heartbeat: Heartbeat)**: boolunivoco
  Metodo per inviare il messaggio di stato.

3.4.2.6. WarehouseStateDTO

1. Rappresenta lo stato del magazzino sotto forma di DTO (Data Transfer Object),
2. Utilizzato per il trasporto dei dati tra livelli dell'applicazione.

Descrizione degli attributi della struttura:

- **state**: string
  Stato del magazzino.

3.4.2.7. WarehouseIdDTO

1. Rappresenta l'identificativo del magazzino sotto forma di DTO.

Descrizione degli attributi della struttura:
- **id**: number
  Identificativo numerico del magazzino.

3.4.2.8. HeartbeatDTO

1. Rappresenta un messaggio di stato del magazzino sotto forma di DTO.

Descrizione degli attributi della struttura:
- **heartbeatMsg**: string
  Contenuto del messaggio di stato.
- **timestamp**: DateTime
  Data e ora del messaggio.
- **warehouseId**: WarehouseIdDTO
  Identificativo del magazzino mittente.

3.4.2.9. DataMapper

1. Gestisce la conversione tra DTO e oggetti di dominio e viceversa.

Può invocare le seguenti funzioni statiche:
- **toDomain(dto: WarehouseStateDTO): WarehouseState**
  Converte il DTO dello stato del magazzino in un oggetto di dominio.
- **toDomain(dto: WarehouseIdDTO): WarehouseId**
  Converte il DTO dell'id del magazzino in un oggetto di dominio.
- **toDTO(domain: WarehouseState): WarehouseStateDTO**
  Converte un oggetto di dominio WarehouseState nel rispettivo DTO.
- **toDTO(domain: WarehouseId): WarehouseIdDTO**
  Converte un oggetto di dominio WarehouseId nel rispettivo DTO.
- **toDTO(domain: Heartbeat): HeartbeatDTO**
  Converte un oggetto di dominio Heartbeat nel rispettivo DTO.

3.4.2.10. GetStateUseCase

1. Casi d'uso per la gestione dello stato del magazzino.

Può invocare le seguenti funzioni:
- **getSyncedState(warehouseIdDTO: WarehouseIdDTO): WarehouseStateDTO**
  Recupera lo stato sincronizzato del magazzino.

3.4.2.11. StateController

1. Gestisce gli eventi in ingresso e coordina i layer sottostanti,
2. Riceve DTO dall'esterno, li converte in oggetti di dominio e invoca il WarehouseStateService.

3.4.2.12. GetStatePort

1. Definisce il contratto per la pubblicazione dello stato del magazzino.

Può invocare le seguenti funzioni:
- **publishState(state: WarehouseState): void**
  Pubblica l'evento di avvenuta pubblicazione dello stato del magazzino.

3.4.2.13. SendHeartBeatPort

1. Definisce il contratto per la pubblicazione dei messaggi di stato (heartbeat).

Può invocare le seguenti funzioni:

- **publishHeartbeat(heartbeat: Heartbeat): void**
  Dichiarazione del metodo che pubblica l'evento di avvenuta pubblicazione del messaggio di stato del magazzino.

3.4.2.14. StateUpdatedPort

1. Definisce il contratto per la notifica di aggiornamenti dello stato del magazzino.

Può invocare le seguenti funzioni:

- **stateUpdated(state: WarehouseState): void**
  Dichiarazione del metodo che pubblica l'evento di avvenuta modifica dello stato del magazzino.

3.4.2.15. StateEventAdapter

1. Gestisce gli eventi in uscita verso altri moduli del sistema,
2. Implementa i metodi di **GetStatePort**, **SendHeartBeatPort** e **StateUpdatedPort**,
3. Converte gli oggetti di dominio nei rispettivi DTO prima della pubblicazione.

3.4.2.16. StateRepository

1. Interfaccia per l'accesso ai dati dello stato del magazzino.

Può invocare le seguenti funzioni:

- **getState(warehouseId: WarehouseId): WarehouseState**
  Restituisce lo stato corrente del magazzino.
- **updateState(state: WarehouseState): boolunivoco**
  Aggiorna lo stato del magazzino e restituisce true/false a seconda del successo dell'operazione.

3.4.2.17. StateRepositoryImpl

3.4.3. Microservizio Cloud State

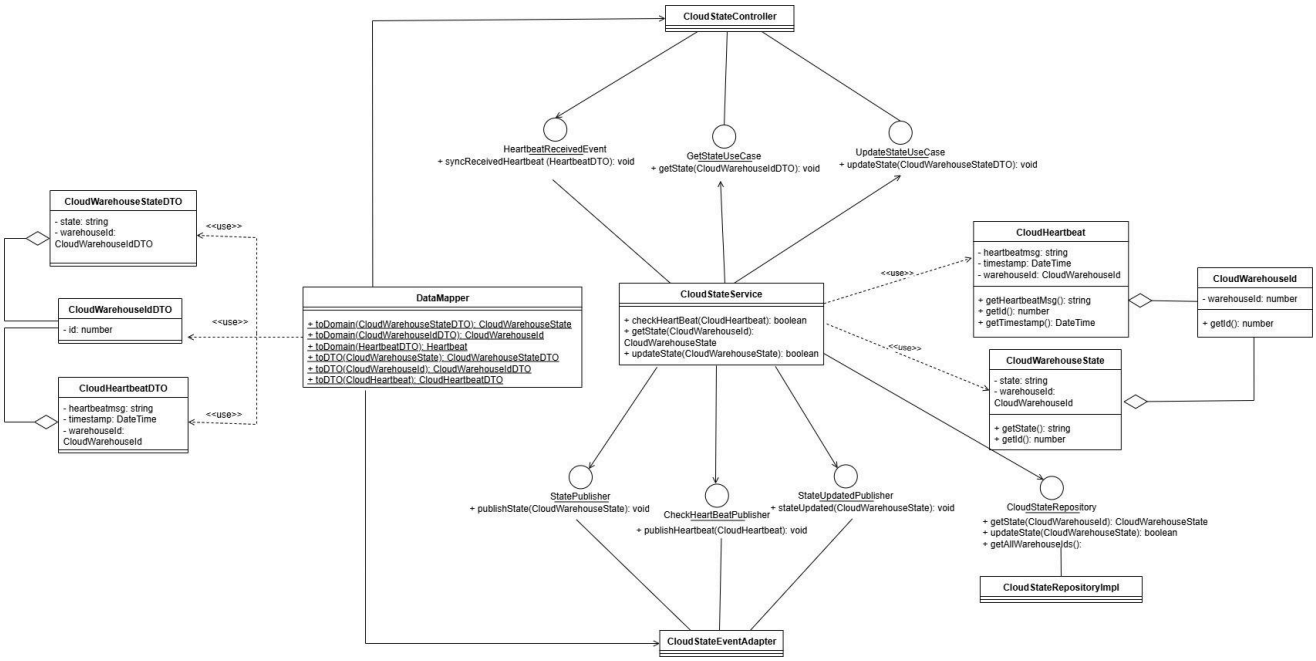

Figura 3: Schema UML - Microservizio Cloud State

3.4.3.1. Descrizione del microservizio

Il **Microservizio Cloud State** rappresenta il componente centralizzato responsabile della raccolta, aggregazione e gestione degli stati provenienti dai singoli magazzini. Il suo compito principale di mandare eventi ad i stati degli warehouse è ricevere i segnali di heartbeat inviati dal **Microservizio Stato**, mantenere una visione coerente e aggiornata della disponibilità complessiva e renderla disponibile agli altri microservizi che necessitano di queste informazioni.

3.4.3.1.1. Funzionalità principali

- **Ricezione heartbeat**: acquisizione dei segnali di stato inviati dai singoli magazzini.
- **Aggregazione stati**: consolidamento delle informazioni per costruire una visione globale del sistema.
- **Aggiornamento stato centralizzato**: mantenimento della coerenza e sincronizzazione degli stati dei magazzini.
- **Consultazione stato aggregato**: esposizione delle informazioni verso microservizi che richiedono lo stato complessivo.
- **Pubblicazione eventi**: invio di notifiche verso i microservizi interessati quando avvengono variazioni di stato.
- **Interoperabilità**: interazione con microservizi esterni quali:
  - *Microservizio Stato*, per ricevere heartbeat e aggiornamenti.
  - *Microservizio Routing*, per ricevere evento per la richiesta dello stato di un magazzino specifico.

3.4.3.2. CloudWarehouseId

1. Rappresenta l'identificatore univoco del magazzino nel cloud,
2. Incapsula il campo *warehouseId: number*,
3. Espone il metodo *getId()*.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  Identificativo numerico del magazzino.

Può invocare le seguenti funzioni:
- **getId()**: number
  Restituisce l'id del magazzino.

3.4.3.3. CloudWarehouseState

1. Contiene lo stato corrente del magazzino nel cloud.

Descrizione degli attributi della struttura:
- **state**: string
  Stato del magazzino.
- **warehouseId**: CloudWarehouseId
  Identificativo del magazzino.

Può invocare le seguenti funzioni:
- **getState()**: string
  Restituisce lo stato del magazzino.
- **getId()**: number
  Restituisce l'id del magazzino.

3.4.3.4. CloudHeartbeat

1. Rappresenta un messaggio periodico di stato inviato dal magazzino al cloud.

Descrizione degli attributi della struttura:
- **heartbeatMsg**: string
  Contenuto del messaggio di stato.
- **timestamp**: DateTime
  Data e ora del messaggio.
- **warehouseId**: CloudWarehouseId
  Identificativo del magazzino mittente.

Può invocare le seguenti funzioni:
- **getHeartbeatMsg()**: string
  Restituisce il messaggio di stato.
- **getId()**: number
  Restituisce l'id del magazzino.
- **getTimestamp()**: DateTime
  Restituisce la data e l'ora del messaggio.

3.4.3.5. CloudStateService

1. Costituisce la logica di business del servizio cloud,
2. Gestisce il controllo, la visualizzazione e l'aggiornamento dello stato del magazzino.

Può invocare le seguenti funzioni:
- **checkHeartbeat(heartbeat: CloudHeartbeat)**: boolunivoco
  Controlla lo stato del magazzino tramite il messaggio di heartbeat.
- **getState(warehouseId: CloudWarehouseId)**: CloudWarehouseState
  Restituisce lo stato del magazzino selezionato.
- **updateState(state: CloudWarehouseState)**: boolunivoco
  Aggiorna lo stato del magazzino.

3.4.3.6. CloudWarehouseStateDTO

1. Rappresenta lo stato del magazzino nel cloud sotto forma di DTO.

Descrizione degli attributi della struttura:

- **state**: string
  Stato del magazzino.
- **warehouseId**: number
  Identificativo del magazzino.

3.4.3.7. CloudWarehouseIdDTO

1. Rappresenta l'identificativo del magazzino sotto forma di DTO.

Descrizione degli attributi della struttura:

- **id**: number
  Identificativo numerico del magazzino.

3.4.3.8. CloudHeartbeatDTO

1. Rappresenta un messaggio di stato del magazzino sotto forma di DTO.

Descrizione degli attributi della struttura:

- **heartbeatMsg**: string
  Contenuto del messaggio di stato.
- **timestamp**: DateTime
  Data e ora del messaggio.
- **warehouseId**: CloudWarehouseId
  Identificativo del magazzino mittente.

3.4.3.9. DataMapper

1. Gestisce la conversione tra DTO e oggetti di dominio e viceversa.

Può invocare le seguenti funzioni statiche:

- **toDomain(dto: CloudWarehouseStateDTO): CloudWarehouseState**
  Converte il DTO dello stato del magazzino in oggetto di dominio.
- **toDomain(dto: CloudWarehouseIdDTO): CloudWarehouseId**
  Converte il DTO dell'id del magazzino in oggetto di dominio.
- **toDTO(domain: CloudWarehouseState): CloudWarehouseStateDTO**
  Converte l'oggetto di dominio CloudWarehouseState nel rispettivo DTO.
- **toDTO(domain: CloudWarehouseId): CloudWarehouseIdDTO**
  Converte l'oggetto di dominio CloudWarehouseId nel rispettivo DTO.
- **toDTO(domain: CloudHeartbeat): CloudHeartbeatDTO**
  Converte l'oggetto di dominio CloudHeartbeat nel rispettivo DTO.

3.4.3.10. GetStateUseCase

1. Caso d'uso per la richiesta dello stato di un magazzino nel cloud.

Può invocare le seguenti funzioni:

- **getState(warehouseIdDTO: CloudWarehouseIdDTO): void**
  Dichiarazione del metodo per richiedere lo stato di un magazzino.

3.4.3.11. HeartbeatReceivedEvent

1. Caso d'uso per la gestione dei messaggi di heartbeat ricevuti.

Può invocare le seguenti funzioni:

- **syncReceivedHeartbeat(heartbeatDTO: CloudHeartbeatDTO): void**
  Sincronizza lo stato del magazzino nel cloud.

3.4.3.12. UpdateStateUseCase

1. Caso d'uso per la sincronizzazione delle modifiche dello stato del magazzino.

Può invocare le seguenti funzioni:
- **updateState(stateDTO: CloudWarehouseStateDTO): void**
  Sincronizza la modifica dello stato del magazzino nel cloud.

3.4.3.13. CloudStateController

1. Gestisce gli eventi in ingresso,
2. Implementa i metodi di **HeartbeatReceivedEvent**, **UpdateStateUseCase** e **GetStateUseCase**,
3. Riceve i DTO dall'esterno, li converte in oggetti di dominio e li passa al **CloudStateService**.

3.4.3.14. GetStatePort

1. Definisce il contratto per la pubblicazione dello stato del magazzino.

Può invocare le seguenti funzioni:
- **publishState(state: CloudWarehouseState): void**
  Pubblica l'evento di avvenuta pubblicazione dello stato del magazzino.

3.4.3.15. CheckHeartBeatPort

1. Definisce il contratto per la pubblicazione dei messaggi di stato (heartbeat).

Può invocare le seguenti funzioni:
- **publishHeartbeat(heartbeat: CloudHeartbeat): void**
  Pubblica l'evento di avvenuta pubblicazione del messaggio di stato del magazzino.

3.4.3.16. StateUpdatedPort

1. Definisce il contratto per la notifica di aggiornamenti dello stato del magazzino.

Può invocare le seguenti funzioni:
- **stateUpdated(state: CloudWarehouseState): void**
  Pubblica l'evento di avvenuta modifica dello stato del magazzino.

3.4.3.17. StateEventAdapter

1. Gestisce gli eventi in uscita verso altri moduli del sistema,
2. Implementa i metodi di **GetStatePort**, **CheckHeartBeatPort** e **StateUpdatedPort**,
3. Converte gli oggetti di dominio nei rispettivi DTO prima della pubblicazione.

3.4.3.18. CloudStateRepository

1. Interfaccia per l'accesso ai dati dello stato del magazzino nel cloud.

Può invocare le seguenti funzioni:
- **getState(warehouseId: CloudWarehouseId): CloudWarehouseState**
  Restituisce lo stato corrente del magazzino.
- **updateState(state: CloudWarehouseState): boolunivoco**
  Aggiorna lo stato del magazzino e restituisce true/false a seconda del successo dell'operazione.

3.4.3.19. CloudStateRepositoryImpl

3.4.4. Microservizio Orders

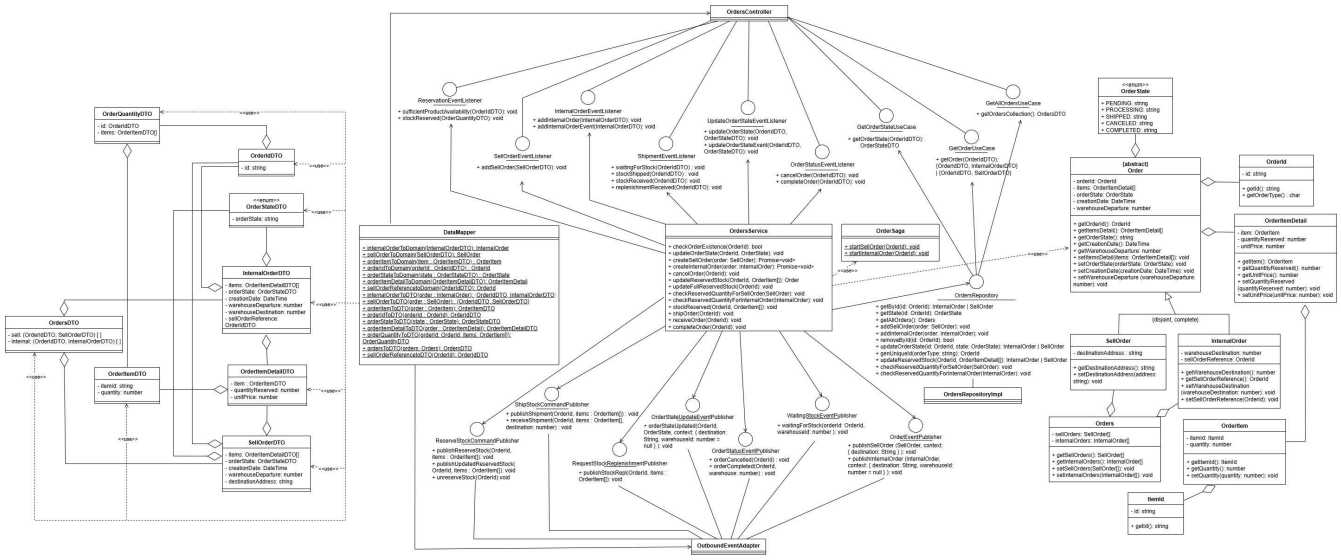

Figura 4: Schema UML - Microservizio Ordini

3.4.4.1. Descrizione del microservizio

Il **Microservizio Ordini** rappresenta il componente responsabile della gestione degli ordini di un magazzino. Il suo compito principale è quello di orchestrare la gestione degli ordini dal momento in cui sono inseriti al momento in cui la merce arriva a destinazione. Si occupa inoltre di comunicare con il sistema centrale per la richiesta di sopperimento merce questa merce non sia sufficiente per soddsfare un ordine.

3.4.4.1.1. Funzionalità principali

- **Gestione degli ordini**: inserimento, annullamento, aggiornamento dello stato e consultazione degli ordini di vendita e degli ordini di trasferimento.
- **Saga degli ordini di vendita**: gestisce la saga per la gestione degli ordini di vendita a partire dal momento in cui sono inseriti. Si occupa in particolare di ordinare:
  - Controllo e riservamento di merce nell'inventario locale.
  - Richiedere merce al sistema centrale qualora fosse necessario.
  - Spedizione della merce.
- **Saga degli ordini di trasferimento**: gestisce la saga per la gestione degli ordini di trasferimento a partire dal momento in cui sono inseriti. Si occupa in particolare di ordinare:
  - Controllo e riservamento di merce nell'inventario locale.
  - Spedizione della merce.
  - Tracciatura dell'ordine anche nel magazzino di destinazione.
- **Pubblicazione eventi**: emissione di notifiche verso gli altri microservizi quando avvengono variazioni di stato dell'ordine.
- **Interoperabilità**: interazione con microservizi esterni quali:
  - *Sistema Centralizzato*, che coordina gli eventi critici e la logica di alto livello.
  - *Microservizio Inventario*, per verificare la disponibilità dei prodotti richiesti.
  - *Ordini Aggregato*, per fornire una vista globale degli ordini nell'intero sistema.
  - *Microservizio Ordini* di altri magazzini, per la gestione di traferimenti interni tra magazzini.

3.4.4.2. OrderId

1. Rappresenta l'identificativo univoco dell'ordine,

2. Incapsula il campo *id: string*,
3. Espone i metodi *getId()* e *getOrderType()*.

Descrizione degli attributi della struttura:
- **id**: string
  Identificativo dell'ordine.

Può invocare le seguenti funzioni:
- **getId()**: string
  Restituisce l'id dell'ordine.
- **getOrderType()**: char
  Restituisce il tipo di ordine (vendita / trasferimento interno), viene differenziato tramite la prima lettera dell'Id (I o V).

3.4.4.3. `<<enum>>` OrderState

1. Rappresenta lo stato dell'ordine.

Descrizione degli attributi della struttura:
- **orderState**: string
  Stato dell'ordine, e lo stato dell'ordine può essere («PENDING», «PROCESSING», «SHIPPED», «CANCELLED» ,«COMPLETED»).

3.4.4.4. ItemId

1. Rappresenta l'identificativo di un prodotto presente nell'ordine.

Descrizione degli attributi della struttura:
- **id**: string
  Identificativo del prodotto.

Può invocare le seguenti funzioni:
- **getId()**: string
  Restituisce l'id del prodotto.

3.4.4.5. OrderItem

1. Rappresenta in maniera sintetica i prodotti ordinati.

Descrizione degli attributi della struttura:
- **itemId**: ItemId
  Identificativo del prodotto.
- **quantity**: number
  Quantità ordinata.

Può invocare le seguenti funzioni:
- **getItemId(): ItemId**
  Restituisce l'id del prodotto ordinato.
- **getQuantity(): number**
  Restituisce la quantità ordinata.
- **setQuantity(quantity: number): void**
  Imposta la quantità ordinata per l'item.

3.4.4.6. OrderItemDetail

1. Rappresenta nel dettaglio i prodotti ordinati.

Descrizione degli attributi della struttura:

- **item**: OrderItem Rappresenta l'id del prodotto e la quantità ordinata.
- **quantityReserved**: number
  Quantità riservata per l'ordine.
- **unitPrice**: number
  Prezzo unitario del prodotto ordinato.

Può invocare le seguenti funzioni:
- **getItemId(): ItemId**
  Restituisce l'id del prodotto.
- **getQuantity(): number**
  Restituisce la quantità ordinata.
- **getQuantityReserved(): number**
  Restituisce la quantità riservata.
- **getUnitPrice(): number**
  Restituisce il prezzo unitario.
- **setQuantity(quantity: number): void**
  Imposta la quantità ordinata.
- **setQuantityReserved(quantityReserved: number): void**
  Imposta la quantità riservata.
- **setUnitPrice(unitPrice: number): void**
  Imposta il prezzo unitario.

3.4.4.7. {abstract} Order

1. Classe astratta per ordini di vendita o interni.
2. È stato scelto questo approccio in quanto ci sono due tipi di ordini: gli ordini di vendita e gli ordini interni.

Descrizione degli attributi della struttura:
- **orderId**: OrderId
  Rappresenta l'id dell'ordine.
- **items**: OrderItemDetail[]
  Rappresenta l'array di prodotti ordinati.
- **orderState**: OrderState
  Rappresenta lo stato dell'ordine.
- **creationDate**: DateTime
  Rappresenta la data e l'ora della creazione dell'ordine.
- **warehouseDeparture**: number
  Rappresenta il magazzino di partenza dell'ordine.

Può invocare le seguenti funzioni:
- **getOrderId(): OrderId**
  Restituisce l'id dell'ordine,
- **getItemsDetail(): OrderItemDetail[]**
  Restituisce l'array di dettaglio dell'ordine dei prodotti.
- **getOrderState(): string**
  Restituisce lo stato dell'ordine.
- **getCreationDate(): DateTime**
  Restituisce la data di creazione.
- **getWarehouseDeparture(): number**
  Restituisce il magazzino di partenza.

- **setItemsDetail(items: OrderItemDetail[]): void**
  Imposta l'array di dettaglio dell'ordine dei prodotti.
- **setOrderState(orderState: OrderState): void**
  Imposta lo stato dell'ordine.
- **setCreationDate(creationDate: DateTime): void**
  Imposta la data di creazione.
- **setWarehouseDeparture(warehouseDeparture: number): void**
  Imposta il magazzino di partenza.

3.4.4.8. SellOrder

1. Classe concreta che rappresenta gli ordini di vendita.

Descrizione degli attributi della struttura:
- **destinationAddress**: string
  Rappresenta l'indirizzo di destinazione dell'ordine.

Può invocare le seguenti funzioni:
- **getDestinationAddress(): string**
  Restituisce l'indirizzo di destinazione.
- **setDestinationAddress(address: string): void**
  Imposta l'indirizzo di destinazione.

3.4.4.9. InternalOrder

1. Classe concreta che rappresenta gli ordini interni, di riassortimento.

Descrizione degli attributi della struttura:
- **warehouseDestination**: number
  Rappresenta il magazzino di destinazione dell'ordine di traferimento.
- **sellOrderReference**: OrderId
  Rappresenta il riferimento ad un ordine di vendita, per i casi di ordini di traferimento richiesti al sistema centrale da un ordine di vendita

Può invocare le seguenti funzioni:
- **getWarehouseDestination(): number**
  Restituisce il magazzino di destinazione.
- **setWarehouseDestination(warehouseDestination: number): void**
  Imposta il magazzino di destinazione.

3.4.4.10. Orders

1. Classe che accorpa tutti gli ordini.
2. È utile, ad esempio, per una visualizzazione complessiva degli ordini.

Descrizione degli attributi della struttura:
- **sellOrders**: SellOrder[] Rappresenta l'array degli ordini di vendita.
- **internalOrders**: InternalOrder[] Rappresenta l'array degli ordini interni.

Può invocare le seguenti funzioni:
- **getSellOrders(): SellOrder[]**
  Restituisce gli ordini di vendita.
- **getInternalOrders(): InternalOrder[]**
  Restituisce gli ordini interni.

- **setSellOrders(sellOrders: SellOrder[]): void**
  Imposta gli ordini di vendita.
- **setInternalOrders(internalOrders: InternalOrder[]): void**
  Imposta gli ordini interni.

3.4.4.11. OrderService

1. Logica di business del microservizio ordini.

Può invocare le seguenti funzioni:

- **checkOrderExistence(OrderId): bool**
  Verifica l'esistenza dell'ordine.
- **updateOrderState(OrderId, OrderState): void**
  Aggiorna lo stato dell'ordine.
- **createSellOrder(SellOrder): Promise**
  Crea un nuovo ordine di vendita.
- **createInternalOrder(order: InternalOrder): Promise**
  Crea un nuovo ordine interno.
- **cancelOrder(OrderId): void**
  Annulla un ordine.
- **updateReservedStock(OrderId, OrderItem[]): Order**
  Aggiorna le quantità riservate dopo il controllo magazzino.
- **updateFullReservedStock(OrderId): Order**
  PRE: Tutti i prodotti richiesti dall'ordine sono presenti in magazzino con le quantità richieste.
  POST: Aggiorna le quantità riservate dopo il controllo magazzino.
- **checkReservedQuantityForSellOrder(SellOrder): void**
  Verifica le quantità riservate per un ordine di vendita.
- **checkReservedQuantityForInternalOrder(InternalOrder): void**
  Verifica le quantità riservate per un ordine interno.
- **shipOrder(OrderId): void**
  Imposta l'ordine come spedito.
- **receiveOrder(OrderId): void**
  Segnala la ricezione dell'ordine.
- **completeOrder(OrderId): void**
  Imposta l'ordine come completato.

3.4.4.12. OrderSaga

1. Gestisce le saghe degli ordini.
2. In seguito verrà spiegata nel dettaglio: per ogni tipologia di ordine (di vendita o di trasferimento interno), c'è un diverso modo di procedere, utilizzando eventi e metodi presenti in questo microservizio.

Può invocare le seguenti funzioni:

- **startSellOrder(orderId: OrderId): void**
  Esegue la saga per un ordine di vendita.
- **startInternalOrder(orderId: OrderId): void**
  Esegue la saga per un ordine interno.

3.4.4.13. OrderIdDTO

1. Rappresenta l'identificativo dell'ordine sotto forma di DTO.

Descrizione degli attributi della struttura:
- **id**: string
  Codice identificativo dell'ordine.

3.4.4.14. <<enum>> OrderStateDTO

1. Rappresenta lo stato dell'ordine sotto forma di DTO.

Descrizione degli attributi della struttura:
- **orderState**: string
  Stato dell'ordine («PENDING», «PROCESSING», «SHIPPED», «CANCELLED» ,«COMPLETED»).

3.4.4.15. OrderItemDTO

1. Rappresenta un prodotto ordinato sotto forma di DTO.

Descrizione degli attributi della struttura:
- **itemId**: string
  Identificativo del prodotto.
- **quantity**: number
  Quantità ordinata.

3.4.4.16. OrderItemDetailDTO

1. Rappresenta nel dettaglio un prodotto ordinato sotto forma di DTO.

Descrizione degli attributi della struttura:
- **item**: OrderItemDTO
  Informazioni sul prodotto ordinato.
- **quantityReserved**: number
  Quantità riservata per l'ordine.
- **unitPrice**: number
  Prezzo unitario del prodotto ordinato.

3.4.4.17. InternalOrderDTO

1. Rappresenta un ordine interno sotto forma di DTO.

Descrizione degli attributi della struttura:
- **orderId**: OrderIdDTO
  Identificativo dell'ordine.
- **items**: OrderItemDetailDTO[]
  Array dei prodotti ordinati.
- **orderState**: OrderStateDTO
  Stato dell'ordine.
- **creationDate**: DateTime
  Data e ora di creazione dell'ordine.
- **warehouseDeparture**: number
  Magazzino di partenza.
- **warehouseDestination**: number
  Magazzino di destinazione.
- **sellOrderId**: OrderIdDTO
  Riferimento a ordine di vendita.

3.4.4.18. SellOrderDTO

1. Rappresenta un ordine di vendita sotto forma di DTO.

Descrizione degli attributi della struttura::

- **orderId**: OrderIdDTO
  Identificativo dell'ordine.
- **items**: OrderItemDetailDTO[]
  Array dei prodotti ordinati.
- **orderState**: OrderStateDTO
  Stato dell'ordine.
- **creationDate**: DateTime
  Data e ora di creazione.
- **warehouseDeparture**: number
  Magazzino di partenza.
- **destinationAddress**: string
  Indirizzo di destinazione dell'ordine.

3.4.4.19. OrderQuantityDTO

1. Rappresenta l'ordine con quantità dei prodotti sotto forma di DTO.

Descrizione degli attributi della struttura:

- **id**: OrderIdDTO
  Identificativo dell'ordine.
- **items**: OrderItemDTO[]
  Array dei prodotti ordinati.

3.4.4.20. OrdersDTO

1. Rappresenta tutti gli ordini aggregati sotto forma di DTO.

Descrizione degli attributi della struttura:

- **sell**: SellOrderDTO[]
  Array degli ordini di vendita.
- **internal**: InternalOrderDTO[]
  Array degli ordini interni.

3.4.4.21. DataMapper

1. Gestisce la conversione tra DTO e oggetti di dominio e viceversa.

Può invocare le seguenti funzioni:

- **internalOrderToDomain(dto: InternalOrderDTO): InternalOrder**
  Converte un DTO InternalOrder in oggetto di dominio.
- **sellOrderToDomain(dto: SellOrderDTO): SellOrder**
  Converte un DTO SellOrder in oggetto di dominio.
- **orderItemToDomain(item: OrderItemDTO): OrderItem**
  Converte un DTO OrderItem in oggetto di dominio.
- **orderIdToDomain(orderId: OrderIdDTO): OrderId**
  Converte un DTO OrderId in oggetto di dominio.
- **orderStateToDomain(state: OrderStateDTO): OrderState**
  Converte un DTO OrderState in oggetto di dominio.
- **orderItemDetailToDomain(dto: OrderItemDetailDTO): OrderItemDetail**
  Converte un DTO OrderItemDetail in oggetto di dominio.

- **internalOrderToDTO(order: InternalOrder): InternalOrderDTO**
  Converte un oggetto di dominio InternalOrder in DTO.
- **sellOrderToDTO(order: SellOrder): SellOrderDTO**
  Converte un oggetto di dominio SellOrder in DTO.
- **orderItemToDTO(order: OrderItem): OrderItemDTO**
  Converte un oggetto di dominio OrderItem in DTO.
- **orderIdToDTO(orderId: OrderId): OrderIdDTO**
  Converte un oggetto di dominio OrderId in DTO.
- **orderStateToDTO(state: OrderState): OrderStateDTO**
  Converte un oggetto di dominio OrderState in DTO.
- **orderItemDetailToDTO(order: OrderItemDetail): OrderItemDetailDTO**
  Converte un oggetto di dominio OrderItemDetail in DTO.
- **orderQuantityToDTO(orderId: OrderId, items: OrderItem[]): OrderQuantityDTO**
  Converte l'OrderId e gli OrderItem in un DTO OrderQuantityDTO.
- **ordersToDTO(orders: Orders): OrdersDTO**
  Converte l'oggetto Orders in DTO aggregato.

3.4.4.22. ReservationEventListener

1. Gestisce gli eventi relativi alla richiesta di riservare stock per un ordine.

Può invocare le seguenti funzioni:

- **sufficientProductAvailability(OrderIdDTO): void**
  Comunica che tutti i prodotti richiesti da un ordine sono presenti nel magazzino con le quanità richieste.
- **stockReserved(order: OrderQuantityDTO): void**
  Comunica la quantità di prodotti riservati dal magazzino per un ordine.

3.4.4.23. SellOrderEventListener

1. Gestisce gli eventi relativi all'aggiunta di ordini di vendita.

Può invocare le seguenti funzioni:

- **addSellOrder(order: SellOrderDTO): void**
  Richiede l'aggiunta di un ordine di vendita.

3.4.4.24. InternalOrderEventListener

1. Gestisce gli eventi relativi all'aggiunta di ordini interni.

Può invocare le seguenti funzioni:

- **addInternalOrder(order: InternalOrderDTO): void**
  Richiede l'aggiunta di un ordine di trasferimento interno.
- **addInternalOrderEvent(order: InternalOrderDTO): void**
  Richiede l'aggiunta di un ordine di trasferimento interno.

3.4.4.25. ShipmentEventListener

1. Gestisce gli eventi di spedizione e ricezione degli ordini.

Può invocare le seguenti funzioni:

- **waitingForStock(orderId: OrderIdDTO): void**
  Comunica al magazzino di partenza che il magazzino di destinazione attende la merce.

- **stockShipped(orderId: OrderIdDTO): void**
  Comunica che il magazzino ha spedito la merce.
- **stockReceived(orderId: OrderIdDTO): void**
  Comunica che il magazzino di destinazione ha ricevuto la merce.
- **replenishmentReceived(orderId: OrderIdDTO): void**
  Comunica che il riassortimento è stato completato.

3.4.4.26. UpdateOrderStateUseCase

1. Gestisce l'aggiornamento dello stato degli ordini.

Può invocare le seguenti funzioni:
- **updateOrderState(orderId: OrderIdDTO, state: OrderStateDTO): void**
  Richiede l'aggiornamento dello stato di un ordine.
- **updateOrderStatEvent(orderId: OrderIdDTO, state: OrderStateDTO): void**
  Richiede l'aggiornamento dello stato di un ordine.

3.4.4.27. OrderStatusEventListener

1. Gestisce gli eventi relativi allo stato finale degli ordini.

Può invocare le seguenti funzioni:
- **cancelOrder(orderId: OrderIdDTO): void**
  Richiede la cancellazione di un ordine.
- **completeOrder(orderId: OrderIdDTO): void**
  Contrassegna l'ordine come completato.

3.4.4.28. GetOrderStateUseCase

1. Gestisce la richiesta di stato di un ordine.

Può invocare le seguenti funzioni:
- **getOrderState(orderId: OrderIdDTO): OrderStateDTO**
  Richiede lo stato corrente dell'ordine.

3.4.4.29. GetOrderUseCase

1. Gestisce la richiesta di visualizzazione di un ordine.

Può invocare le seguenti funzioni:
- **getOrder(orderId: OrderIdDTO): InternalOrderDTO | SellOrderDTO**
  Richiede i dettagli di un ordine specifico.

3.4.4.30. GetAllOrderUseCase

1. Gestisce la richiesta di visualizzazione di tutti gli ordini.

Può invocare le seguenti funzioni:
- **getOrdersCollection(): OrdersDTO**
  Richiede la lista completa degli ordini.

3.4.4.31. OrdersController

1. Gestisce gli eventi in ingresso e coordina i DTO verso il servizio.

Può invocare le seguenti funzioni:
- Implementa i metodi di
  - **ReservationEventListener**,
  - **SellOrderEventListener**,

- **InternalOrderEventListener**,
- **ShipmentEventListener**,
- **UpdateOrderStateUseCase**,
- **OrderStatusEventListener**,
- **GetOrderStateUseCase**,
- **GetOrderUseCase**,
- **GetAllOrderUseCase**.

- Converte i DTO in oggetti di dominio e li passa a OrdersService.

3.4.4.32. ReserveStockCommandPublisher

1. Gestisce la pubblicazione di eventi per riservare stock.

Può invocare le seguenti funzioni:

- **publishReserveStock(orderId: OrderId, items: OrderItem[]): void**
  Pubblica l'evento di richiesta di riserva prodotti dal magazzino.
- **publishUpdatedReserveStock(orderId: OrderId, items: OrderItem[]): void**
  Pubblica l'evento di richiesta di aggiornare i prodotti riservati.
- **unservedStock(OrderId) : void**
  Pubblica l'evento di mancata disponibilità dei prodotti richiesti perciò richiede di annullare le quantità riservate fin ora.

3.4.4.33. ShipStockCommandPublisher

1. Gestisce la pubblicazione degli eventi di spedizione.

Può invocare le seguenti funzioni:

- **publishShipment(orderId: OrderId, items: OrderItem[]): void**
  Pubblica l'evento di spedizione della merce.
- **receiveShipment(orderId: OrderId, items: OrderItem[], destination: number): void**
  Pubblica l'evento di ricezione della merce da parte del magazzino di destinazione.

3.4.4.34. RequestStockReplenishmentPublisher

1. Gestisce la pubblicazione degli eventi di riassortimento.

Può invocare le seguenti funzioni:

- **publishStockRepl(orderId: OrderId, items: OrderItem[]): void**
  Pubblica l'evento di riassortimento dello stock.

3.4.4.35. OrderUpdateEventPublisher

1. Gestisce la pubblicazione degli eventi di aggiornamento degli ordini.

Può invocare le seguenti funzioni:

- **orderStateUpdated(order: Order): void**
  Pubblica l'evento di aggiornamento di un ordine.

3.4.4.36. OrderStatusEventPublisher

1. Gestisce la pubblicazione degli eventi relativi allo stato finale degli ordini.

Può invocare le seguenti funzioni:

- **orderCancelled(orderId: OrderId, warehouse: number): void**
  Pubblica l'evento di cancellazione di un ordine.

- **orderCompleted(orderId: OrderId, warehouse: number): void**
  Pubblica l'evento di completamento di un ordine.

3.4.4.37. OrderEventPublisher

1. Gestisce la pubblicazione degli ordini interni e di vendita.

Può invocare le seguenti funzioni:

- **publishInternalOrder (InternalOrder, context: { destination: String, warehouseId: number = null }): void**
  Pubblica l'ordine interno.
- **publishSellOrder (SellOrder, context: { destination: String, warehouseId: Integer = null }): void**
  Pubblica l'ordine di vendita.

3.4.4.38. WaitingStockEventPublisher

1. Gestisce la pubblicazione degli eventi di attesa della merce.

Può invocare le seguenti funzioni:

- **waitingForStock(orderId: OrderId, warehouseId: number): void**
  Pubblica l'ordine di vendita.

3.4.4.39. OutboundEventAdapter

1. Gestisce gli eventi in uscita.

Può invocare le seguenti funzioni:

- Implementa i metodi di:
  - ReserveStockCommandPublisher,
  - ShipStockCommandPublisher,
  - RequestStockReplenishmentPublisher,
  - OrderUpdateEventPublisher,
  - OrderStatusEventPublisher,
  - OrderEventPublisher,
  - WaitingStockEventPublisher.
- Converte gli oggetti di dominio nei rispettivi DTO.

3.4.4.40. OrdersRepository

1. Gestisce l'accesso e la persistenza degli ordini.

Può invocare le seguenti funzioni:

- **getById(id: OrderId): InternalOrder | SellOrder**
  Restituisce un ordine tramite id.
- **getState(id: OrderId): OrderState**
  Restituisce lo stato di un ordine.
- **getAllOrders(): Orders**
  Restituisce tutti gli ordini.
- **addSellOrder(order: SellOrder): void**
  Aggiunge un ordine di vendita.
- **addInternalOrder(order: InternalOrder): void**
  Aggiunge un ordine interno.
- **removeById(id: OrderId): bool**
  Rimuove un ordine e restituisce true/false.

- **genUniqueId(orderType: string): OrderId**
  Genera un id univoco per un ordine.
- **updateOrderState(id: OrderId, state: OrderState): InternalOrder | SellOrder**
  Aggiorna lo stato di un ordine.
- **updateReservedStock(orderId: OrderId, items: OrderItem[]): InternalOrder | SellOrder**
  Aggiorna le quantità riservate per un ordine.
- **checkReservedQuantityForSellOrder(SellOrder): void**
  Verifica se le quantità di prodotti riservate per l'ordine di vendita corrispondono a quelle richieste
- **checkReservedQuantityForInternalOrder(SellOrder): void**
  Verifica se le quantità di prodotti riservate per l'ordine di trasferiemento corrispondono a quelle richieste

3.4.4.41. OrdersRepositoryImpl

3.4.5. Microservizio Ordine Aggregato

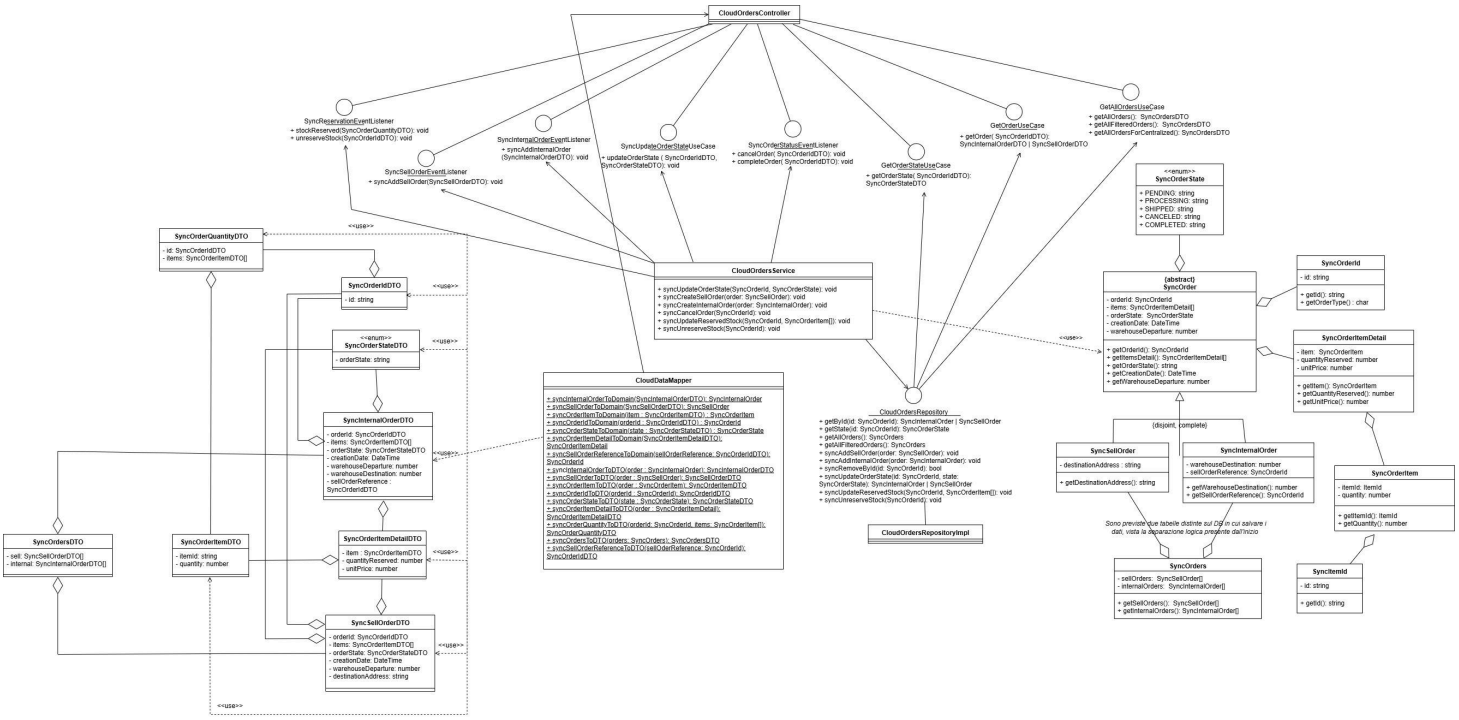

Figura 5: Schema UML - Microservizio Ordine Aggregato

3.4.5.1. Descrizione del microservizio

Il Microservizio Ordini invia eventi al Microservizio Ordini Aggregato, che ha il compito di:

- Ricevere e consolidare dati sugli ordini provenienti da più magazzini.
- Offrire una vista centralizzata dello stato degli ordini.
- Garantire consistenza e tracciabilità nel caso di trasferimenti tra magazzini differenti.
- Offrire dati al sistema centralizzato per risolvere problemi di criticità.

3.4.5.1.1. Funzionalità principali

- **Aggregazione ordini**: riceve e consolida dati sugli ordini da più magazzini.
- **Gestione dati** : Manda i dati richiesti dal sistema centralizzato per supportarlo al compimento del processo.

3.4.5.2. SyncOrderId

1. Rappresenta l'identificativo dell'ordine sincronizzato.

Descrizione degli attributi della struttura:

- **id**: string
  Codice identificativo dell'ordine.

Può invocare le seguenti funzioni:

- **getId(): string**
  Restituisce l'id dell'ordine.
- **getOrderType(): char**
  Restituisce il tipo di ordine (di vendita / trasferimento interno).

3.4.5.3. `<<enum>>` SyncOrderState

1. Rappresenta lo stato dell'ordine sincronizzato.

Descrizione degli attributi della struttura:

- **orderState**: string
  Stato dell'ordine («PENDING», «PROCESSING», «SHIPPED», «CANCELLED» ,«COMPLETED»).

3.4.5.4. SyncItemId

1. Rappresenta l'identificativo del prodotto presente nell'ordine sincronizzato.

Descrizione degli attributi della struttura:
- **id**: string
  Codice identificativo del prodotto.

Può invocare le seguenti funzioni:
- **getId(): string**
  Restituisce l'id del prodotto.

3.4.5.5. SyncOrderItem

1. Rappresenta un prodotto ordinato in forma semplificata.

Descrizione degli attributi della struttura:
- **itemId**: SyncItemId
  Identificativo del prodotto.
- **quantity**: number
  Quantità ordinata.

Può invocare le seguenti funzioni:
- **getItemId(): SyncItemId**
  Restituisce l'id del prodotto.
- **getQuantity(): number**
  Restituisce la quantità ordinata.

3.4.5.6. SyncOrderItemDetail

1. Rappresenta un prodotto ordinato in forma dettagliata.

Descrizione degli attributi della struttura:
- **item**: SyncOrderItem
  Informazioni sul prodotto ordinato.
- **quantityReserved**: number
  Quantità riservata per l'ordine.
- **unitPrice**: number
  Prezzo unitario del prodotto ordinato.

Può invocare le seguenti funzioni:
- **getItemId(): SyncItemId**
  Restituisce l'id del prodotto.
- **getQuantity(): number**
  Restituisce la quantità ordinata.
- **getQuantityReserved(): number**
  Restituisce la quantità riservata.
- **getUnitPrice(): number**
  Restituisce il prezzo unitario.

3.4.5.7. {abstract} SyncOrder

1. Classe astratta per gli ordini sincronizzati (vendita e interni).

Descrizione degli attributi della struttura:
- **orderId**: SyncOrderId
  Identificativo dell'ordine.
- **items**: SyncOrderItemDetail[]
  Array dei prodotti ordinati.
- **orderState**: SyncOrderState
  Stato dell'ordine.
- **creationDate**: DateTime
  Data e ora di creazione.
- **warehouseDeparture**: number
  Magazzino di partenza.

Può invocare le seguenti funzioni:
- **getItems(): SyncOrderItem[]**
  Restituisce l'array dei prodotti ordinati.
- **getOrderState(): string**
  Restituisce lo stato dell'ordine.
- **getCreationDate(): DateTime**
  Restituisce la data di creazione.
- **getWarehouseDeparture(): number**
  Restituisce il magazzino di partenza.

3.4.5.8. SyncSellOrder

1. Rappresenta un ordine di vendita sincronizzato.

Descrizione degli attributi della struttura:
- **destinationAddress**: string
  Indirizzo di destinazione.

Può invocare le seguenti funzioni:
- **getDestinationAddress(): string**
  Restituisce l'indirizzo di destinazione.

3.4.5.9. SyncInternalOrder

1. Rappresenta un ordine interno sincronizzato (riassortimento).

Descrizione degli attributi della struttura:
- **warehouseDestination**: number
  Magazzino di destinazione.

Può invocare le seguenti funzioni:
- **getWarehouseDestination(): number**
  Restituisce il magazzino di destinazione.

3.4.5.10. SyncOrders

1. Accorpa tutti gli ordini sincronizzati.

Descrizione degli attributi della struttura:
- **sellOrders**: SyncSellOrder[]
  Array degli ordini di vendita.
- **internalOrders**: SyncInternalOrder[]
  Array degli ordini interni.

Può invocare le seguenti funzioni:
- **getSellOrders(): SyncSellOrder[]**
  Restituisce l'array degli ordini di vendita.
- **getInternalOrders(): SyncInternalOrder[]**
  Restituisce l'array degli ordini interni.

3.4.5.11. CloudOrderService

1. Logica di business del microservizio Aggregate Orders.

Può invocare le seguenti funzioni:
- **syncUpdateOrderState(orderId: SyncOrderId, state: SyncOrderState): void**
  Sincronizza l'aggiornamento dello stato di un ordine.
- **syncCreateSellOrder(order: SyncSellOrder): Promise<void>**
  Sincronizza la creazione di un ordine di vendita.
- **syncCreateInternalOrder(order: SyncInternalOrder): Promise<void>**
  Sincronizza la creazione di un ordine interno.
- **syncCancelOrder(orderId: SyncOrderId): void**
  Sincronizza la cancellazione di un ordine.
- **syncUpdateReservedStock(orderId: SyncOrderId, items: SyncOrderItem[]): SyncOrder**
  Sincronizza l'aggiornamento delle quantità riservate.
- **syncUnreserveStock(SyncOrderId): void**
  Sincronizza la cancellazione delle quantità riservate.

3.4.5.12. SyncOrderIdDTO

1. DTO per l'id dell'ordine sincronizzato.

Descrizione degli attributi della struttura:
- **id**: string
  Codice identificativo dell'ordine.

3.4.5.13. <<enum>> SyncOrderStateDTO

1. DTO per lo stato dell'ordine sincronizzato.

Descrizione degli attributi della struttura:
- **orderState**: string
  Stato dell'ordine.

3.4.5.14. SyncOrderItemDTO

1. DTO per un prodotto ordinato.

Descrizione degli attributi della struttura:
- **itemId**: string
  Id del prodotto.
- **quantity**: number
  Quantità ordinata.

3.4.5.15. SyncOrderItemDetailDTO

1. DTO per un prodotto ordinato in dettaglio.

Descrizione degli attributi della struttura:

- **item**: SyncOrderItemDTO
  Id e quantità del prodotto.
- **quantityReserved**: number
  Quantità riservata.
- **unitPrice**: number
  Prezzo unitario.

3.4.5.16. SyncInternalOrderDTO

1. DTO per un ordine interno sincronizzato.

Descrizione degli attributi della struttura:

- **orderId**: SyncOrderIdDTO
  Id dell'ordine.
- **items**: SyncOrderItemDetailDTO[]
  Prodotti ordinati.
- **orderState**: SyncOrderStateDTO
  Stato dell'ordine.
- **creationDate**: DateTime
  Data di creazione.
- **warehouseDeparture**: number
  Magazzino di partenza.
- **warehouseDestination**: number
  Magazzino di destinazione.

3.4.5.17. SyncSellOrderDTO

1. DTO per un ordine di vendita sincronizzato.

Descrizione degli attributi della struttura:

- **orderId**: SyncOrderIdDTO
  Id dell'ordine.
- **items**: SyncOrderItemDetailDTO[]
  Prodotti ordinati.
- **orderState**: SyncOrderStateDTO
  Stato dell'ordine.
- **creationDate**: DateTime
  Data di creazione.
- **warehouseDeparture**: number
  Magazzino di partenza.
- **destinationAddress**: string
  Indirizzo di destinazione.

3.4.5.18. SyncOrderQuantityDTO

1. DTO per quantità dei prodotti ordinati.

Descrizione degli attributi della struttura:

- **id**: SyncOrderIdDTO
  Id dell'ordine.
- **items**: SyncOrderItemDTO[]
  Array dei prodotti ordinati.

3.4.5.19. SyncOrdersDTO

1. DTO aggregato per tutti gli ordini sincronizzati.

Descrizione degli attributi della struttura:

- **sell**: SyncSellOrderDTO[]
  Array degli ordini di vendita.
- **internal**: SyncInternalOrderDTO[]
  Array degli ordini interni.

3.4.5.20. DataMapper

1. Gestisce la conversione tra DTO e oggetti di dominio per gli ordini sincronizzati.

Può invocare le seguenti funzioni:

- **syncInternalOrderToDomain(dto: SyncInternalOrderDTO)** : SyncInternalOrder
  Converte un DTO di ordine interno in oggetto di dominio.
- **syncSellOrderToDomain(dto: SyncSellOrderDTO)** : SyncSellOrder
  Converte un DTO di ordine di vendita in oggetto di dominio.
- **syncOrderItemToDomain(item: SyncOrderItemDTO)** : SyncOrderItem
  Converte un DTO di prodotto in oggetto di dominio.
- **syncOrderIdToDomain(orderId: SyncOrderIdDTO)** : SyncOrderId
  Converte un DTO dell'id ordine in oggetto di dominio.
- **syncOrderStateToDomain(state: SyncOrderStateDTO)** : SyncOrderState
  Converte un DTO dello stato ordine in oggetto di dominio.
- **syncOrderItemDetailtoDomain(dto: SyncOrderItemDetailDTO)**: SyncOrderItemDetail
  Converte un DTO di prodotto dettagliato in oggetto di dominio.
- **syncInternalOrderToDTO(order: SyncInternalOrder)** : SyncInternalOrderDTO
  Converte un oggetto di dominio InternalOrder in DTO.
- **syncSellOrderToDTO(order: SyncSellOrder)** : SyncSellOrderDTO
  Converte un oggetto di dominio SellOrder in DTO.
- **syncOrderItemToDTO(item: SyncOrderItem)** : SyncOrderItemDTO
  Converte un oggetto di dominio OrderItem in DTO.
- **syncOrderIdToDTO(orderId: SyncOrderId)** : SyncOrderIdDTO
  Converte un oggetto di dominio OrderId in DTO.
- **syncOrderStateToDTO(state: SyncOrderState)** : SyncOrderStateDTO
  Converte un oggetto di dominio OrderState in DTO.
- **syncOrderItemDetailToDTO(order: SyncOrderItemDetail)** : SyncOrderItemDetailDTO
  Converte un oggetto di dominio OrderItemDetail in DTO.
- **syncOrderQuantityToDTO(orderId: SyncOrderId, items: SyncOrderItem[])** : SyncOrderQuantityDTO
  Converte l'insieme OrderId + prodotti in DTO SyncOrderQuantityDTO.
- **syncOrdersToDTO(orders: SyncOrders)** : SyncOrdersDTO
  Converte l'oggetto SyncOrders in DTO aggregato.

3.4.5.21. SyncReservationEventListener

1. Gestisce gli eventi in ingresso relativi alla riservazione dei prodotti.

Può invocare le seguenti funzioni:

- **stockReserved(dto: SyncOrderQuantityDTO)** : void
  Richiede la sincronizzazione delle quantità di prodotti riservati per un ordine.
- **unreserveStock(SyncOrderIdDTO)** : void
  Richiede la sincronizzazione della cancellazione delle quantità riservate.

3.4.5.22. SyncSellOrderEventListener

1. Gestisce gli eventi in ingresso relativi agli ordini di vendita.

Può invocare le seguenti funzioni:
- **addSellOrder(dto: SyncSellOrderDTO)** : void
  Richiede la sincronizzazione dell'aggiunta di un ordine di vendita.

3.4.5.23. SyncInternalOrderEventListener

1. Gestisce gli eventi in ingresso relativi agli ordini interni.

Può invocare le seguenti funzioni:
- **addInternalOrder(dto: SyncInternalOrderDTO)** : void
  Richiede la sincronizzazione dell'aggiunta di un ordine di trasferimento interno.

3.4.5.24. SyncUpdateOrderStateUseCase

1. Gestisce gli eventi in ingresso per aggiornamento dello stato degli ordini.

Può invocare le seguenti funzioni:
- **updateOrderState(orderIdDTO: SyncOrderIdDTO, stateDTO: SyncOrderStateDTO)** : void
  Richiede la sincronizzazione dell'aggiornamento dello stato dell'ordine.

3.4.5.25. SyncOrderStatusEventListener

1. Gestisce gli eventi in ingresso relativi allo stato finale dell'ordine.

Può invocare le seguenti funzioni:
- **cancelOrder(orderIdDTO: SyncOrderIdDTO)** : void
  Richiede la sincronizzazione della cancellazione dell'ordine.
- **completeOrder(orderIdDTO: SyncOrderIdDTO)** : void
  Contrassegna lo stato dell'ordine come completato.

3.4.5.26. GetOrderStateUseCase

1. Permette di richiedere lo stato di un ordine.

Può invocare le seguenti funzioni:
- **getOrderState(orderIdDTO: SyncOrderIdDTO)** : SyncOrderStateDTO
  Restituisce lo stato dell'ordine specificato.

3.4.5.27. GetOrderUseCase

1. Permette di richiedere la visualizzazione di un ordine specifico.

Può invocare le seguenti funzioni:
- **getOrder(orderIdDTO: SyncOrderIdDTO)** : SyncInternalOrderDTO | SyncSellOrderDTO
  Restituisce l'ordine interno o di vendita corrispondente.

3.4.5.28. GetAllOrderUseCase

1. Permette di richiedere la visualizzazione di tutti gli ordini.

Può invocare le seguenti funzioni:
- **getAllOrders()** : SyncOrdersDTO
  Restituisce tutti gli ordini sincronizzati.
- **getAllFilteredOrders(state: SyncOrderStateDTO)** : SyncOrdersDTO
  Restituisce tutti gli ordini sincronizzati filtrati per stato.
- **getAllOrdersForCentralized()** : SyncOrdersDTO
  Restituisce tutti gli ordini sincronizzati senza filtri, per il sistema centralizzato.

3.4.5.29. CloudOrdersController

1. Gestisce tutti gli eventi in ingresso relativi agli ordini sincronizzati.

Descrizione:
- Implementa i metodi di SyncReservationEventListener, SyncSellOrderEventListener, SyncInternalOrderEventListener, SyncUpdateOrderStateUseCase, SyncOrderStatusEventListener, GetOrderStateUseCase, GetOrderUseCase, GetAllOrderUseCase.
- Riceve i DTO dall'esterno, li converte in oggetti di dominio e li passa al CloudOrderService.

3.4.5.30. CloudOrdersRepository

1. Gestisce la persistenza e l'accesso ai dati degli ordini sincronizzati.

Può invocare le seguenti funzioni:
- **getById(id: SyncOrderId)** : SyncInternalOrder | SyncSellOrder
  Restituisce l'ordine corrispondente all'id.
- **getState(id: SyncOrderId)** : SyncOrderState
  Restituisce lo stato dell'ordine.
- **getAllOrders()** : SyncOrders
  Restituisce tutti gli ordini.
- **getAllFilteredOrders()** : SyncOrders
  Restituisce tutti gli ordini filtrati.
- **syncAddSellOrder(order: SyncSellOrder)** : void
  Sincronizza l'aggiunta di un ordine di vendita.
- **syncAddInternalOrder(order: SyncInternalOrder)** : void
  Sincronizza l'aggiunta di un ordine interno.
- **syncRemoveById(id: SyncOrderId)** : bool
  Sincronizza la rimozione di un ordine, fornendo true/false alla rimozione.
- **syncUpdateOrderState(id: SyncOrderId, state: SyncOrderState)** : SyncInternalOrder | SyncSellOrder
  Sincronizza l'aggiornamento dello stato di un ordine.
- **syncUpdateReservedStock(orderId: SyncOrderId, items: SyncOrderItem[])** : SyncInternalOrder | SyncSellOrder
  Sincronizza l'aggiornamento delle quantità riservate.
- **syncUnreserveStock(SyncOrderId)** : void
  Sincronizza la cancellazione delle quantità riservate.

3.4.5.31. CloudOrdersRepositoryImpl

3.4.6. Microservizio Inventario Aggregato

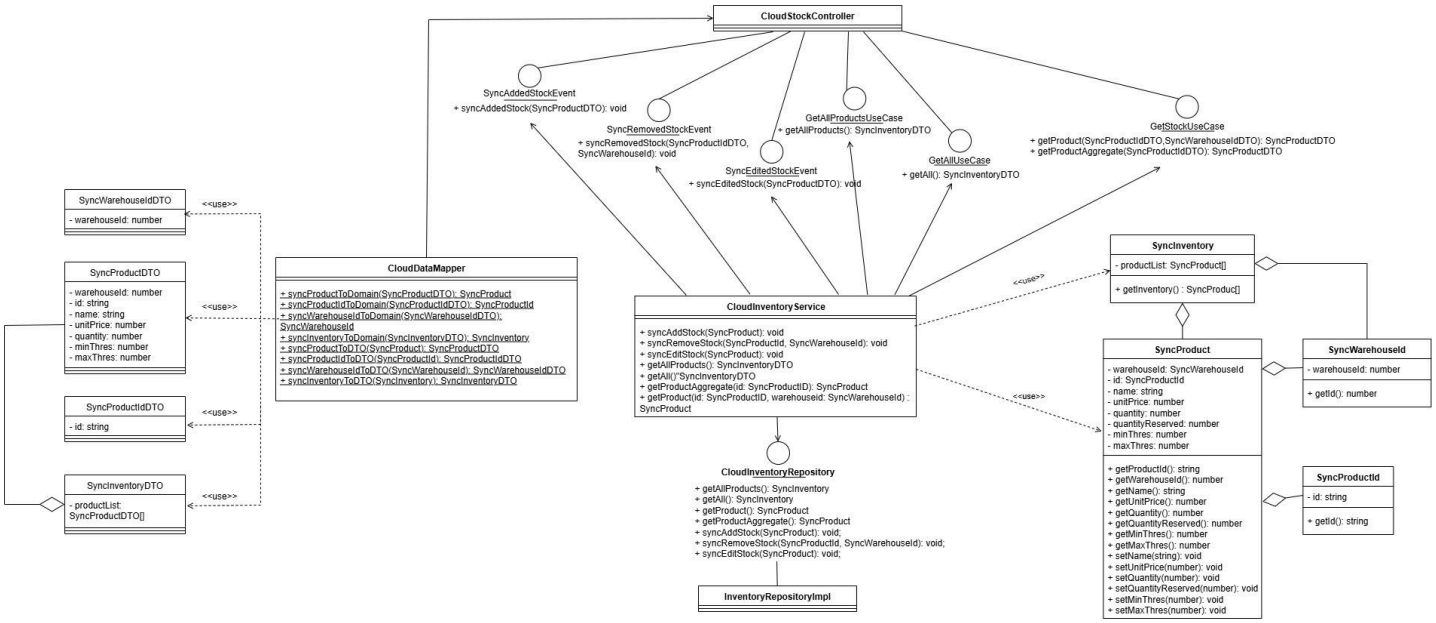

Figura 6: Schema UML - Microservizio Inventario Aggregato

3.4.6.1. Descrizione del microservizio

Il Microservizio invia eventi al Microservizio Inventario Aggregato, che ha il compito di:

- Ricevere e consolidare dati sui prodotti provenienti da più magazzini.
- Offrire una vista centralizzata dello stato dell'inventario.
- Garantire la quantità tra magazzini differenti al sistema centralizzato.

3.4.6.1.1. Funzionalità principali

- **Aggregazione Inventario** : riceve e consolida dati su prodotti da più magazzini.
- **Gestione dati** : Manda i dati richiesti dal sistema centralizzato per supportarlo al compimento del processo.

3.4.6.2. SyncProduct

1. Rappresenta un prodotto in un magazzino sincronizzato.

Descrizione degli attributi della struttura:

- **id**: string
  Codice identificativo del prodotto.
- **WarehouseId**: number
  Codice identificativo del warehouse.
- **name**: string
  Nome del prodotto.
- **uniPrice**: number
  Prezzo unitario del prodotto.
- **quantity**: number
  Quantità disponibile del prodotto.
- **minThres**: number
  Soglia minima di riordino.
- **maxThres**: number
  Soglia massima di riordino.

Può invocare le seguenti funzioni:

- **getProductId()**: string
  Ottiene l'id del prodotto.
- **getWarehouseId()**: number
  Ottiene l'id del magazzino.
- **getName()**: string
  Ottiene il nome del prodotto.
- **getUnitPrice()**: number
  Ottiene il prezzo unitario del prodotto.
- **getQuantity()**: number
  Ottiene la quantità disponibile del prodotto.
- **getQuantityReserved()**: number
  Ottiene la quantità riservata del prodotto.
- **getMinThres()**: number
  Ottiene la soglia minima di un prodotto.
- **getMaxThres()**: number
  Ottiene la soglia massima di un prodotto.
- **setName(string)**: void
  setta il nome del prodotto.
- **setUnitPrice(number)**: void
  setta il prezzo unitario del prodotto.
- **setQuantity(number)**: void
  setta la quantità disponibile del prodotto.
- **setQuantityReserved(number)**: void
  setta la quantità riservata del prodotto.
- **setMinThres(number)**: void
  setta la soglia minima di un prodotto.
- **setMaxThres(number)**: void
  setta la soglia massima di un prodotto.

3.4.6.3. SyncProductId

1. Rappresenta l'identificativo del prodotto sincronizzato.

Descrizione degli attributi della struttura:

- **id**: string
  Codice identificativo del prodotto.

Può invocare le seguenti funzioni:

- **getId(): string**
  Restituisce l'id del prodotto.

3.4.6.4. SyncWarehouseId

1. Rappresenta l'identificativo del magazzino sincronizzato.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  Codice identificativo del magazzino.

Può invocare le seguenti funzioni:

- **getId(): number**
  Restituisce l'id del magazzino.

3.4.6.5. SyncInventory

1. Rappresenta l'inventario di un magazzino sincronizzato.

Descrizione degli attributi della struttura:

- **productList : SyncProduct[]**
  Array dei prodotti presenti nel magazzino.

Può invocare le seguenti funzioni:

- **getInventory() : SyncProduct[]**
  Restituisce l'array dei prodotti presenti nel magazzino.

3.4.6.6. CloudInventoryService

1. Logica di business del microservizio Aggregate Inventory.

Può invocare le seguenti funzioni:

- **syncAddStock(SyncProduct)**: void
  Effettua la sincronizzazione dell'aggiunta di un prodotto facendo la chiamata del metodo sulla repository.
- **syncRemoveStock(SyncProductId, SyncWarehouseId)**: void
  Effettua la sincronizzazione della rimozione di un prodotto facendo la chiamata del metodo sulla repository.
- **syncEditStock(SyncProduct)**: void Effettua la sincronizzazione della modifica di un prodotto facendo la chiamata del metodo sulla repository.
- **getAllProducts()**: SyncInventoryDTO Effettua la sincronizzazione dell'ottenimento di tutti i prodotti aggregati facendo la chiamata del metodo sulla repository.
- **getAll()**: SyncInventoryDTO Effettua la sincronizzazione dell'ottenimento di tutti i prodotti in generale per ognuno il magazzino di appartenenza facendo la chiamata del metodo sulla repository.
- **getProductAggregate(id: SyncProductID)**: SyncProduct Effettua la sincronizzazione dell'ottenimento di un prodotto aggregato tramite id facendo la chiamata del metodo sulla repository.
- **getProduct(id: SyncProductID, warehouseid: SyncWarehouseId)** : SyncProduct Effettua la sincronizzazione dell'ottenimento di un prodotto tramite id e magazzino facendo la chiamata del metodo sulla repository.

3.4.6.7. SyncProductDTO

1. Rappresenta un dto in un magazzino sincronizzato.

Descrizione degli attributi della struttura:

- **id**: string
  Codice identificativo del prodotto in DTO.
- **WarehouseId**: number
  Codice identificativo del warehouse in DTO.
- **name**: string
  Nome del prodotto in DTO.
- **uniPrice**: number
  Prezzo unitario del prodotto in DTO.
- **quantity**: number
  Quantità disponibile del prodotto in DTO.
- **minThres**: number
  Soglia minima di riordino in DTO.

- **maxThres**: number
  Soglia massima di riordino in DTO.

3.4.6.8. SyncProductIdDTO

1. Rappresenta l'identificativo del prodotto sincronizzato in DTO.

Descrizione degli attributi della struttura:
- **id**: string
  Codice identificativo del prodotto.

3.4.6.9. SyncWarehouseIdDTO

1. Rappresenta l'identificativo del magazzino sincronizzato in DTO.

Descrizione degli attributi della struttura:
- **warehouseId**: number
  Codice identificativo del magazzino in DTO.

3.4.6.10. SyncInventoryDTO

1. Rappresenta l'inventario di un magazzino sincronizzato in DTO.

Descrizione degli attributi della struttura:
- **productList : SyncProduct**[]
  Array dei prodotti presenti nel magazzino in DTO.

3.4.6.11. DataMapper

1. Gestisce la conversione tra DTO e oggetti di dominio per gli ordini sincronizzati.

3.5. Funzioni di sincronizzazione Inventory/Product

Può invocare le seguenti funzioni:
- **syncProductToDomain(dto: SyncProductDTO)** : SyncProduct
  Converte un DTO di prodotto in oggetto di dominio.
- **syncProductIdToDomain(dto: SyncProductIdDTO)** : SyncProductId
  Converte un DTO di identificativo prodotto in oggetto di dominio.
- **syncWarehouseIdToDomain(dto: SyncWarehouseIdDTO)** : SyncWarehouseId
  Converte un DTO di identificativo magazzino in oggetto di dominio.
- **syncInventoryToDomain(dto: SyncInventoryDTO)** : SyncInventory
  Converte un DTO di inventario in oggetto di dominio.
- **syncProductToDTO(product: SyncProduct)** : SyncProductDTO
  Converte un oggetto di dominio Product in DTO.
- **syncProductIdToDTO(productId: SyncProductId)** : SyncProductIdDTO
  Converte un oggetto di dominio ProductId in DTO.
- **syncWarehouseIdToDTO(warehouseId: SyncWarehouseId)** : SyncWarehouseIdDTO
  Converte un oggetto di dominio WarehouseId in DTO.
- **syncInventoryToDTO(inventory: SyncInventory)** : SyncInventoryDTO
  Converte un oggetto di dominio Inventory in DTO.

3.5.0.1. CloudStockController

1. Gestisce tutti gli eventi in ingresso relativi ai prodotti sincronizzati.

Può invocare le seguenti funzioni:

- **syncAddedStock(SyncProductDTO)**: void
  Metodo che andrà a chiamare la sincronizza l'aggiunta di un prodotto effettuando un altra chiamata metodo del service.
- **syncRemovedStock(SyncProductIdDTO, SyncWarehouseIdDTO)**: void
  Metodo che andrà a chiamare la sincronizza la rimozione di un prodotto effettuando un altra chiamata metodo del service.
- **syncEditedStock(SyncProductDTO)**: void
  Metodo che andrà a chiamare lasincronizza la modifica di un prodotto effettuando un altra chiamata metodo del service.
- **getAllProducts()**: SyncInventoryDTO
  Metodo che andrà ad ottenere una lista di tutti i prodotti aggregati effettuando un altra chiamata metodo del service.
- **getAll()**: SyncInventoryDTO
  Metodo che andrà ad ottenere una lista di tutti i prodotti in generale per ognuno il magazzino di appartenenza effettuando un altra chiamata metodo del service.
- **getProductAggregate(id: SyncProductIDDTO)**: SyncProductDTO
  Metodo che andrà a ricavare un prodotto aggregato tramite id effettuando un altra chiamata metodo del service.
- **getProduct(id: SyncProductIDDTO, warehouseid: SyncWarehouseIdDTO)** : SyncProductDTO
  Metodo che andrà a ricavare un prodotto tramite id e magazzino effettuando un altra chiamata metodo del service.

3.5.0.2. SyncAddedStockEvent

1. Gestisce gli eventi in ingresso relativi all'aggiunta di un prodotto per la sincronizzazione effettuando un altra chiamata metodo del service.

Può invocare le seguenti funzioni:

- **syncAddedStock(SyncProductDTO)**: void
  sincronizza l'aggiunta di un prodotto.

3.5.0.3. SyncRemovedStockEvent

1. Gestisce gli eventi in ingresso relativi alla rimozione di un prodotto per la sincronizzazione.

Può invocare le seguenti funzioni:

- **syncRemovedStock(SyncProductIdDTO, SyncWarehouseIdDTO)**: void
  sincronizza la rimozione di un prodotto.

3.5.0.4. SyncEditedStockEvent

1. Gestisce gli eventi in ingresso relativi alla modifica di un prodotto per la sincronizzazione.

Può invocare le seguenti funzioni:

- **syncEditedStock(SyncProductDTO)**: void
  sincronizza la modifica di un prodotto.

3.5.0.5. getAllProductsUsecase

1. Permette di richiedere la visualizzazione di tutti i prodotti aggregati.

Può invocare le seguenti funzioni:

- **getAllProducts()**: SyncInventoryDTO
  ottiene una lista di tutti i prodotti aggregati.

3.5.0.6. GetAllUseCase

1. Permette di richiedere la visualizzazione di tutti i prodotti in generale per ognuno il magazzino di appartenenza.

Può invocare le seguenti funzioni:

- **getOrderState(orderIdDTO: SyncOrderIdDTO)** : SyncOrderStateDTO
  Restituisce lo stato dell'ordine specificato.

3.5.0.7. GetOrderUseCase

1. Permette di richiedere la visualizzazione di un ordine specifico.

- **getAll()**: SyncInventoryDTO
  ottiene una lista di tutti i prodotti in generale per ognuno il magazzino di appartenenza.

3.5.0.8. GetStockUseCase

1. Permette di richiedere la visualizzazione di un prodotto specifico.

Può invocare le seguenti funzioni:

- **getProductAggregate(id: SyncProductIDDTO)**: SyncProductDTO
  ottiene un prodotto aggregato tramite id.
- **getProduct(id: SyncProductIDDTO, warehouseid: SyncWarehouseIdDTO)** : SyncProductDTO
  ottiene un prodotto tramite id e magazzino.

3.5.0.9. CloudInventoryRepository

1. Gestisce la persistenza e l'accesso ai dati degli ordini sincronizzati.

Può invocare le seguenti funzioni:

- **getAllProducts()** : SyncInventory
  Ottieni la lista di tutti i prodotti aggregati.
- **getAll()** : SyncInventory
  Ottieni la lista di tutti i prodotti in generale per ognuno il magazzino di appartenenza.
- **getProduct()** : SyncProduct
  Ottiene un prodotto tramite id e magazzino.
- **getProductAggregate()** : SyncProduct
  Ottiene un prodotto aggregato tramite id.
- **syncAddStock(SyncProduct)** : void
  aggiunge un prodotto.
- **syncRemoveStock(SyncProductId, SyncWarehouseId)** : void
  rimuove un prodotto.
- **syncEditStock(SyncProduct)** : void
  modifica un prodotto.

3.5.0.10. InventoryRepositoryImpl

3.5.1. Sistema Centrale

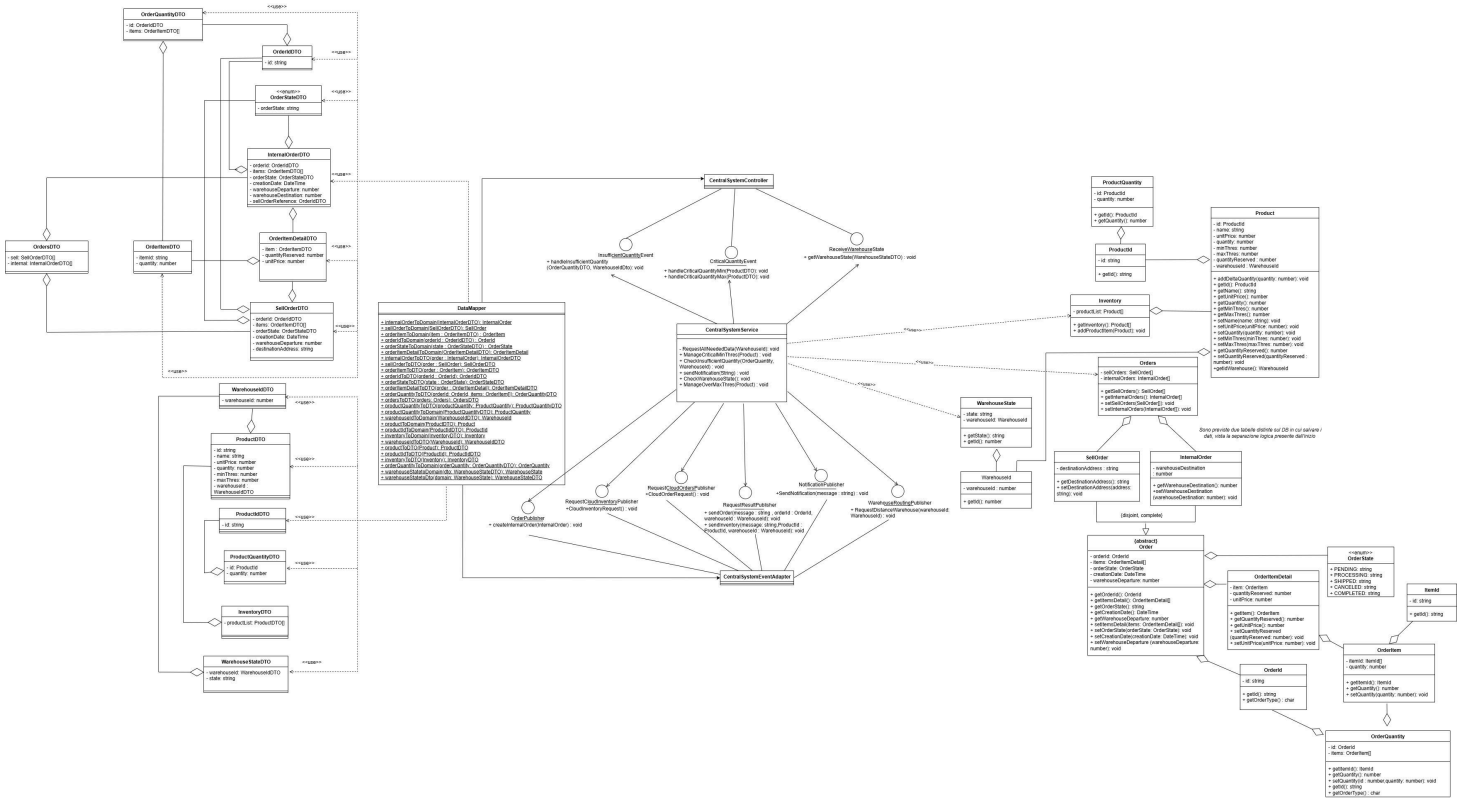

Figura 7: Schema UML - Microservizio Sistema Centralizzato

3.5.1.1. Descrizione del microservizio

Il **Sistema Centralizzato** rappresenta il componente core dell'architettura, responsabile del coordinamento delle operazioni e della gestione integrata delle informazioni provenienti dai diversi microservizi. Il suo compito principale è orchestrare i flussi legati agli ordini, monitorare lo stato dei magazzini e gestire i casi di criticità, come giacenze sotto soglia o indisponibilità operative.

3.5.1.1.1. Funzionalità principali

- **Gestione soglie minime**: monitoraggio degli eventi emessi dagli inventari in caso di criticità sulle quantità di uno o più prodotti e applicazione di azioni correttive.
- **Gestione ordini critici**: elaborazione e instradamento delle richieste d'ordine con prodotti insufficienti, valutando magazzini alternativi (priorità a quelli più vicini) e integrando dati di inventario e routing.
- **Aggregazione dati (Inventari e Ordini)**: consolidamento delle informazioni di giacenza e ordini dai diversi magazzini, a supporto del bilanciamento delle soglie e della gestione delle criticità.
- **Coordinamento stati operativi**: utilizzo delle informazioni del Cloud State per garantire disponibilità e operatività dei magazzini.
- **Routing e strategia di instradamento**: valutazione delle distanze tra magazzini tramite il microservizio Routing e definizione della sequenza ottimale per bilanciamento delle soglie per inventari o criticità degli ordini.
- **Pubblicazione eventi**: emissione di notifiche verso i microservizi interessati quando vengono rilevate criticità o applicate strategie di bilanciamento.

- **Interoperabilità**: interazione con microservizi esterni (Cloud State, Routing, Inventario Aggregato, Ordine Aggregato) per integrare dati e decisioni.

3.5.1.2. CentralSystemController

1. Gestisce tutti gli eventi in ingresso dal cloud e dai magazzini.

Può invocare le seguenti funzioni:

- **CloudOrder(dto: OrderQuantityDTO)** : void
  Riceve un evento di ordine aggregato (OrdersDTO) dal microservizio Orders.
- **CloudInventory(dto: InventoryDTO)** : void
  Riceve un evento di inventario aggregato (InventoryDTO) dal microservizio Inventory.
- **handleInsufficientQuantity(dto: OrderQuantityDTO)** : void
  Gestisce gli ordini per cui il magazzino destinato non ha quantità sufficiente.
- **handleCriticalQuantity(dto: ProductQuantityDTO, warehouseIdDTO: WarehouseIdDTO)** : void
  Gestisce le situazioni critiche di scorta minima dei prodotti in magazzino.
- **getWarehouseDistanceArr(dto: WarehouseStateDTO)** : void
  Riceve le distanze tra il magazzino gestito e tutti gli altri magazzini.
- **getWarehouseState(dto: WarehouseStateDTO)** : void
  Riceve lo stato di un magazzino (online/offline).

3.5.1.3. CentralSystemService

1. Contiene la logica di business del sistema centralizzato.

Descrizione degli attributi della struttura:

- **order** : Order[]
  Lista degli ordini ricevuti.
- **inventory** : Inventory[]
  Lista degli inventari gestiti.
- **warehouseId** : WarehouseId
  Identificativo del magazzino.
- **warehouseState** : WarehouseState[]
  Lista degli stati dei magazzini.
- **orderQuantity** : OrderQuantity
  Contiene le quantità degli ordini.
- **distance** : WarehouseState[]
  Contiene informazioni sulle distanze tra i magazzini.
- **idInternalOrderCounter** : number Contiene informazioni sull'ultimo id arrivato per ordini interni
- **idSellOrderCounter** : number Contiene informazioni sull'ultimo id arrivato per ordini di vendita

Metodi:

- **RequestAllNeededData()** : void
  Richiede dati a Inventory Aggregato, Orders Aggregato e Routing.
- **checkRestocking(order, inventory, distance, productQuantity)** : void
  Il metodo gestisce la problematica del criticalQuantity. Esso avendo distance possiamo determinare la distanza dei magazzini partendo da quello più vicino, verificando se ciascun magazzino dispone di una quantità sufficiente del prodotto per soddisfare la richiesta.

Appena un magazzino dà la conferma di disponibilità, verrà selezionato quel magazzino. Contestualmente, il metodo esegue un controllo sugli ordini presenti per il magazzino. Se esiste un ordine relativo allo stesso prodotto, viene sottratta la quantità impegnata dagli ordini alla disponibilità effettiva del prodotto, verificando quindi se la quantità residua è ancora sufficiente a soddisfare la richiesta.

- **CheckInsufficientQuantity(order, inventory, distance, orderQuantity)** : void
  Il metodo gestisce la problematica del InsufficientQuantity. Esso avendo distance possiamo determinare la distanza dei magazzini partendo da quello più vicino, verificando se ciascun magazzino dispone di una quantità sufficiente del prodotto per soddisfare la richiesta. Appena un magazzino dà la conferma di disponibilità, verrà selezionato quel magazzino. Contestualmente, il metodo esegue un controllo sugli ordini presenti per il magazzino. Se esiste un ordine relativo allo stesso prodotto, viene sottratta la quantità impegnata dagli ordini alla disponibilità effettiva del prodotto, verificando quindi se la quantità residua è ancora sufficiente a soddisfare la richiesta.
- **sendNotification(message: string)** : void
  Invia notifiche a entità interessate.

3.5.1.4. CentralSystemEventAdapter

1. Gestisce la pubblicazione degli eventi verso i microservizi esterni.

Può invocare le seguenti funzioni:

- **createInternalOrder(order: InternalOrder)** : void
  Pubblica un ordine interno verso il microservizio corrispondente.
- **CloudInventoryRequest(inventory: Inventory)** : void
  Richiede report inventario dal microservizio aggregato.
- **CloudOrderRequest(order: Order)** : void
  Richiede report ordini dal microservizio aggregato.
- **RequestWarehouseState(id: WarehouseId)** : void
  Richiede lo stato di un magazzino.
- **SendNotification(message: string)** : void
  Pubblica notifiche a più enti interessati.
- **RequestDistanceWarehouse(warehouseId: WarehouseId)** : void
  Richiede al microservizio Routing la distanza tra il magazzino passato e tutti gli altri.

3.5.1.5. DataMapper

1. Esegue la traduzione tra entità di dominio e DTO,
2. Mantiene l'isolamento tra livello applicativo e dominio,
3. Garantisce la consistenza tra modelli interni e rappresentazioni esterne.

Descrizione dei metodi principali:

- **internalOrderToDomain(order: InternalOrderDTO)** : InternalOrder
  Converte un DTO di ordine interno in entità InternalOrder.
- **sellOrderToDomain(order: SellOrderDTO)** : SellOrder
  Converte un DTO di ordine di vendita in entità SellOrder.

- **orderItemToDomain(item: OrderItemDTO)** : OrderItem
  Converte un DTO di item d'ordine in OrderItem.
- **orderIdToDomain(orderId: OrderIdDTO)** : OrderId
  Converte un identificativo d'ordine DTO in OrderId.
- **orderStateToDomain(state: OrderStateDTO)** : OrderState
  Converte un DTO di stato in enum OrderState.
- **orderItemDetailToDomain(item: OrderItemDetailDTO)** : OrderItemDetail
  Converte un DTO con dettaglio item in entità di dominio.
- **productQuantityToDomain(pq: ProductQuantityDTO)** : ProductQuantity
  Converte un DTO di quantità prodotto in entità ProductQuantity.
- **warehouseIdToDomain(warehouseId: WarehouseIdDTO)** : WarehouseId
  Converte un DTO di magazzino in WarehouseId.
- **productToDomain(product: ProductDTO)** : Product
  Converte un DTO prodotto in Product.
- **productIdToDomain(productId: ProductIdDTO)** : ProductId
  Converte un DTO identificativo in ProductId.
- **inventoryToDomain(inventory: InventoryDTO)** : Inventory
  Converte un inventario DTO in entità Inventory.
- **internalOrderToDTO(order: InternalOrder)** : InternalOrderDTO
  Serializza un InternalOrder in DTO.
- **sellOrderToDTO(order: SellOrder)** : SellOrderDTO
  Serializza un SellOrder in DTO.
- **orderItemToDTO(item: OrderItem)** : OrderItemDTO
  Serializza un OrderItem in DTO.
- **orderIdToDTO(orderId: OrderId)** : OrderIdDTO
  Serializza un OrderId in DTO.
- **orderStateToDTO(state: OrderState)** : OrderStateDTO
  Serializza uno stato ordine in DTO.
- **orderItemDetailToDTO(item: OrderItemDetail)** : OrderItemDetailDTO
  Serializza un dettaglio item ordine in DTO.
- **orderQuantityToDTO(orderId: OrderId, items: OrderItem[])** : OrderQuantityDTO
  Costruisce un DTO di quantità ordini da entità di dominio.
- **ordersToDTO(orders: Orders)** : OrdersDTO
  Serializza un aggregato Orders in DTO.
- **productQuantityToDTO(pq: ProductQuantity)** : ProductQuantityDTO
  Serializza un ProductQuantity in DTO.
- **warehouseIdToDTO(warehouseId: WarehouseId)** : WarehouseIdDTO
  Serializza un WarehouseId in DTO.
- **productToDTO(product: Product)** : ProductDTO
  Serializza un Product in DTO.

- **productIdToDTO(productId: ProductId)** : ProductIdDTO
  Serializza un ProductId in DTO.
- **inventoryToDTO(inventory: Inventory)** : InventoryDTO
  Serializza un inventario in DTO.

3.5.1.6. OrderQuantityDTO

1. Rappresenta la quantità di articoli relativi a un ordine specifico,
2. Utilizzato per trasmettere dati tra livello applicativo e dominio.

Descrizione degli attributi:
- **id**: OrderIdDTO
  Identificativo univoco dell'ordine.
- **items**: OrderItemDTO[]
  Collezione di item associati all'ordine.

3.5.1.7. OrdersDTO

1. Contiene la rappresentazione serializzata di tutti gli ordini,
2. Distingue tra ordini di vendita e ordini interni.

Descrizione degli attributi della struttura:
- **sell**: SellOrderDTO[]
  Lista di ordini di vendita.
- **internal**: InternalOrderDTO[]
  Lista di ordini interni.

3.5.1.8. OrderItemDTO

1. Identifica un singolo articolo presente in un ordine,
2. Trasporta informazioni essenziali per quantità e riferimento.

Descrizione degli attributi della struttura:
- **itemId**: string
  Identificativo univoco dell'oggetto.
- **quantity**: number
  Quantità dell'oggetto ordinata.

3.5.1.9. OrderIdDTO

1. Incapsula l'identificativo univoco di un ordine nella versione DTO.

Descrizione degli attributi della struttura:
- **id**: string
  Identificativo univoco dell'ordine.

3.5.1.10. OrderStateDTO `<<enum>>`

1. Rappresenta lo stato di un ordine nella versione DTO.

Descrizione degli attributi della struttura:
- **orderState**: string
  Indica lo stato attuale (pending, processing, ecc.).

3.5.1.11. OrderItemDetailDTO

1. Fornisce i dettagli di un item d'ordine con quantità e prezzo unitario.

Descrizione degli attributi della struttura:

- **orderItem**: OrderItemDTO
  Oggetto ordine di riferimento.
- **productReserved**: number
  Quantità riservata per il prodotto.
- **unitPrice**: number
  Prezzo unitario del prodotto.

3.5.1.12. SellOrderDTO

1. DTO per rappresentare un ordine di vendita.

Descrizione degli attributi della struttura:

- **orderId**: OrderIdDTO
  Identificativo dell'ordine.
- **orderItem**: OrderItemDTO[]
  Prodotti inclusi nell'ordine.
- **orderState**: OrderStateDTO
  Stato dell'ordine.
- **destinationAddress**: string
  Indirizzo di destinazione.
- **creationDate**: DateTime
  Data di creazione.
- **warehouseDeparture**: number
  Magazzino di partenza.

3.5.1.13. InternalOrderDTO

1. DTO per rappresentare un ordine interno.

Descrizione degli attributi della struttura:

- **orderId**: OrderIdDTO
  Identificativo dell'ordine.
- **orderItem**: OrderItemDTO[]
  Prodotti inclusi nell'ordine.
- **orderState**: OrderStateDTO
  Stato dell'ordine.
- **warehouseDeparture**: number
  Magazzino di partenza.
- **warehouseDestination**: number
  Magazzino di destinazione.
- **sellOrderReference**: OrderIdDTO
  Riferimento a ordine di vendita.

3.5.1.14. WarehouseIdDTO

1. Identificativo del magazzino nella versione DTO.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  Id numerico del magazzino.

3.5.1.15. ProductDTO

1. Rappresentazione serializzata di un prodotto,
2. Trasporta le informazioni di base per comunicazione tra servizi.

Descrizione degli attributi della struttura:
- **id**: string
  Codice univoco del prodotto.
- **name**: string
  Nome del prodotto.
- **price**: number
  Prezzo unitario.
- **maxThres**: number
  Quantità massima consentita.
- **minThres**: number
  Quantità minima consentita.

3.5.1.16. ProductQuantityDTO

1. Associa un prodotto alla sua quantità in forma DTO.

Descrizione degli attributi della struttura:
- **id**: ProductIdDTO
  Identificativo del prodotto.
- **quantity**: number
  Quantità associata.

3.5.1.17. ProductIdDTO

1. Incapsula l'identificativo univoco di un prodotto (univoco) in forma DTO.

Descrizione degli attributi della struttura:
- **id**: string
  Identificativo univoco del prodotto.

3.5.1.18. InventoryDTO

1. Contiene la collezione di prodotti di un magazzino in forma DTO.

Descrizione degli attributi della struttura:
- **productList**: ProductDTO[]
  Lista di prodotti con i relativi attributi.

3.5.1.19. WarehouseStateDTO

1. Rappresenta lo stato di un magazzino (online/offline).

Descrizione degli attributi della struttura:
- **warehouseId**: WarehouseDTO
  Identificativo del magazzino.
- **state**: string
  Stato operativo del magazzino.

3.5.1.20. ProductQuantity

1. Entità che lega un prodotto ad una quantità,
2. Utilizzata negli ordini e nella gestione scorte.

Descrizione degli attributi della struttura:
- **id**: ProductId
  Identificativo prodotto.

- **quantity**: number
  Quantità associata.

Può invocare le seguenti funzioni:
- **getId()**: ProductId
  Restituisce l'identificativo del prodotto.
- **getQuantity()**: number
  Restituisce la quantità disponibile/associata.

3.5.1.21. Product

1. Entità di dominio che incapsula i dati e comportamenti di un prodotto,
2. Gestisce aspetti quantitativi e soglie di sicurezza.

Descrizione degli attributi della struttura:
- **id**: ProductId
  Identificativo univoco del prodotto (univoco).
- **name**: string
  Nome del prodotto.
- **unitPrice**: number
  Prezzo unitario.
- **quantity**: number
  Quantità attualmente disponibile.
- **maxThres**: number
  Soglia massima di disponibilità.
- **minThres**: number
  Soglia minima di disponibilità.
- **quantityReserved**: number
  Quantità riservata.

Può invocare le seguenti funzioni:
- **addDeltaQuantity(quantity: number)**: void
  Aggiorna la quantità disponibile incrementandola o decrementandola.
- **getId()**: ProductId
  Restituisce l'identificativo del prodotto.
- **getName()**: string
  Restituisce il nome del prodotto.
- **getUnitPrice()**: number
  Restituisce il prezzo unitario.
- **getQuantity()**: number
  Restituisce la quantità disponibile.
- **getMinThres()**: number
  Restituisce la soglia minima di scorta.
- **getMaxThres()**: number
  Restituisce la soglia massima di scorta.
- **getQuantityReserved()**: number
  Restituisce la quantità riservata.
- **setName(name: string)**: void
  Aggiorna il nome del prodotto.
- **setUnitPrice(unitPrice: number)**: void
  Aggiorna il prezzo unitario.

- **setQuantity(quantity: number)**: void
  Imposta la quantità disponibile.
- **setMinThres(minThres: number)**: void
  Imposta la soglia minima.
- **setMaxThres(maxThres: number)**: void
  Imposta la soglia massima.
- **setQuantityReserved(quantityReserved: number)**: void
  Imposta la quantità riservata.

3.5.1.22. ProductId

1. Value Object che incapsula l'identificativo univoco di un prodotto,
2. Garantisce consistenza logica e comparabilità.

Descrizione degli attributi della struttura:

- **id**: string
  Identificativo univoco del prodotto.

Può invocare le seguenti funzioni:

- **getId()**: string
  Restituisce il valore dell'identificativo.

3.5.1.23. Orders

1. Aggregato che rappresenta l'insieme degli ordini del sistema,
2. Comprende ordini interni e di vendita.

Descrizione degli attributi della struttura:

- **sellOrders**: SellOrder[]
  Lista ordini di vendita.
- **internalOrders**: InternalOrder[]
  Lista ordini interni.

Può invocare le seguenti funzioni:

- **getSellOrders()**: SellOrder[]
  Restituisce la lista ordini di vendita.
- **getInternalOrders()**: InternalOrder[]
  Restituisce la lista ordini interni.
- **setSellOrder(order: SellOrder)**: void
  Imposta un ordine di vendita.
- **setInternalOrder(order: InternalOrder)**: void
  Imposta un ordine interno.

3.5.1.24. Inventory

1. Collezione di prodotti gestiti da un magazzino,
2. Permette di recuperare o aggiungere articoli all'inventario.

Descrizione degli attributi della struttura:

- **productList**: Product[]
  Lista dei prodotti gestiti.

Può invocare le seguenti funzioni:

- **getInventory(productId: ProductId): Product**
  Restituisce il prodotto corrispondente all'ID.

- **addProductItem(product: Product)**: void
  Aggiunge un prodotto all'inventario.

3.5.1.25. OrderId

1. Oggetto che incapsula l'identificativo univoco di un ordine,
2. Assicura consistenza e comparabilità.

Descrizione degli attributi della struttura:

- **id: string**
  Identificativo univoco dell'ordine.

Può invocare le seguenti funzioni:

- **getId()**: string
  Restituisce l'identificativo dell'ordine.
- **getOrderType()**: char
  Restituisce il tipo di ordine (vendita o interno).

3.5.1.26. OrderState `<<enum>>`

1. Enum che rappresenta lo stato di avanzamento di un ordine,
2. Permette di tracciare l'intero ciclo di vita.

Valori possibili:

- **PENDING**: Ordine in attesa.
- **PROCESSING**: Ordine in elaborazione.
- **SHIPPING**: Ordine in spedizione.
- **CANCELLED**: Ordine annullato.
- **COMPLETED**: Ordine completato.

3.5.1.27. ItemId

1. Rappresenta il Value Object che incapsula l'identificativo univoco di un item (riga d'ordine o riferimento a risorsa). Garantisce confronti di uguaglianza affidabili e coerenza nella serializzazione/deserializzazione.

Descrizione degli attributi della struttura:

- **id: string**
  Identificativo univoco dell'item.

Può invocare le seguenti funzioni:

- **getId(): string**
  Restituisce l'identificativo univoco dell'item.

3.5.1.28. OrderItem

1. Rappresenta un singolo prodotto presente in un ordine e la quantità associata.

Descrizione degli attributi della struttura:

- **itemId**: ItemId[]
  Lista dei prodotti contenuti nell'ordine.
- **quantity**: number
  Quantità ordinata del prodotto.

Può invocare le seguenti funzioni:

- **getItemId()**: ItemId
  Restituisce la lista dei prodotti presenti nell'ordine.

- **getQuantity()**: number
  Restituisce la quantità del prodotto.
- **setQuantity(quantity: number)**: void
  Imposta la quantità del prodotto.

3.5.1.29. OrderItemDetail

1. Rappresenta il dettaglio di un item di un ordine, includendo il prodotto, la quantità riservata e il prezzo unitario.

Descrizione degli attributi della struttura:
- **item**: OrderItem
  L'item d'ordine associato.
- **quantityReserved**: number
  La quantità riservata di quel prodotto per l'ordine.
- **unitPrice**: number
  Prezzo unitario del prodotto.

Può invocare le seguenti funzioni:
- **getItem()**: OrderItem
  Restituisce l'item d'ordine.
- **getQuantityReserved()**: number
  Restituisce la quantità riservata per l'item.
- **getUnitPrice()**: number
  Restituisce il prezzo unitario del prodotto.
- **setQuantityReserved(quantityReserved: number)**: void
  Imposta la quantità riservata dell'item.
- **setUnitPrice(unitPrice: number)**: void
  Imposta il prezzo unitario dell'item.

3.5.1.30. Order (abstract)

1. Rappresenta l'entità astratta comune a tutti i tipi di ordine, gestendo gli attributi fondamentali come ID, lista degli item, stato, data di creazione e magazzino di partenza.

Descrizione degli attributi della struttura:
- **orderId**: OrderId
  Identificativo univoco dell'ordine.
- **items**: OrderItemDetail[]
  Lista dei dettagli dei prodotti contenuti nell'ordine.
- **orderState**: OrderState
  Stato corrente dell'ordine (Pending, Processing, Shipping, Cancelled, Completed).
- **creationDate**: DateTime
  Data e ora di creazione dell'ordine.
- **warehouseDeparture**: number
  ID del magazzino di partenza dell'ordine.

Può invocare le seguenti funzioni:
- **getOrderId()**: OrderId
  Restituisce l'identificativo dell'ordine.
- **getItemsDetail()**: OrderItemDetail[]
  Restituisce la lista dettagliata degli item dell'ordine.

- **getOrderState()**: string
  Restituisce lo stato corrente dell'ordine.
- **getCreationDate()**: DateTime
  Restituisce la data di creazione dell'ordine.
- **getWarehouseDeparture()**: number
  Restituisce il magazzino di partenza.
- **setItemsDetail(items: OrderItemDetail[])**: void
  Aggiorna la lista degli item dell'ordine.
- **setOrderState(orderState: OrderState)**: void
  Aggiorna lo stato dell'ordine.
- **setCreationDate(creationDate: DateTime)**: void
  Imposta la data di creazione dell'ordine.
- **setWarehouseDeparture(warehouseDeparture: number)**: void
  Imposta il magazzino di partenza dell'ordine.

3.5.1.31. SellOrder

1. Rappresenta un ordine di vendita effettuato da un cliente, con indirizzo di destinazione e dettagli dei prodotti ordinati.

Descrizione degli attributi della struttura:
- **destinationAddress**: string
  Indirizzo di spedizione dell'ordine.

Può invocare le seguenti funzioni:
- **getDestinationAddress()**: string
  Restituisce l'indirizzo di destinazione.
- **setDestinationAddress(address: string)**: void
  Imposta l'indirizzo di destinazione dell'ordine.

3.5.1.32. InternalOrder

1. Rappresenta un ordine interno di trasferimento di prodotti tra magazzini.

Descrizione degli attributi della struttura:
- **warehouseDestination**: number
  ID del magazzino di destinazione.

Può invocare le seguenti funzioni:
- **getWarehouseDestination()**: number
  Restituisce il magazzino di destinazione.
- **setWarehouseDestination(warehouseDestination: number)**: void
  Imposta il magazzino di destinazione.

3.5.1.33. WarehouseId

1. Rappresenta un magazzino fisico all'interno del sistema.

Descrizione degli attributi della struttura:
- **warehouseId**: number
  Identificativo univoco del magazzino.

Può invocare le seguenti funzioni:
- **getWarehouseId()**: number
  Restituisce l'ID del magazzino.

3.5.1.34. WarehouseState

1. Rappresenta lo stato operativo di un magazzino, utile per determinare la disponibilità di stock.

Descrizione degli attributi della struttura:

- **state**: string
  Stato operativo del magazzino (es. online/offline).
- **warehouseId**: WarehouseId
  Magazzino a cui si riferisce lo stato.

Può invocare le seguenti funzioni:

- **getState()**: string
  Restituisce lo stato del magazzino.
- **getId()**: number
  Restituisce l'ID del magazzino associato

3.5.2. Microservizio Autenticazione

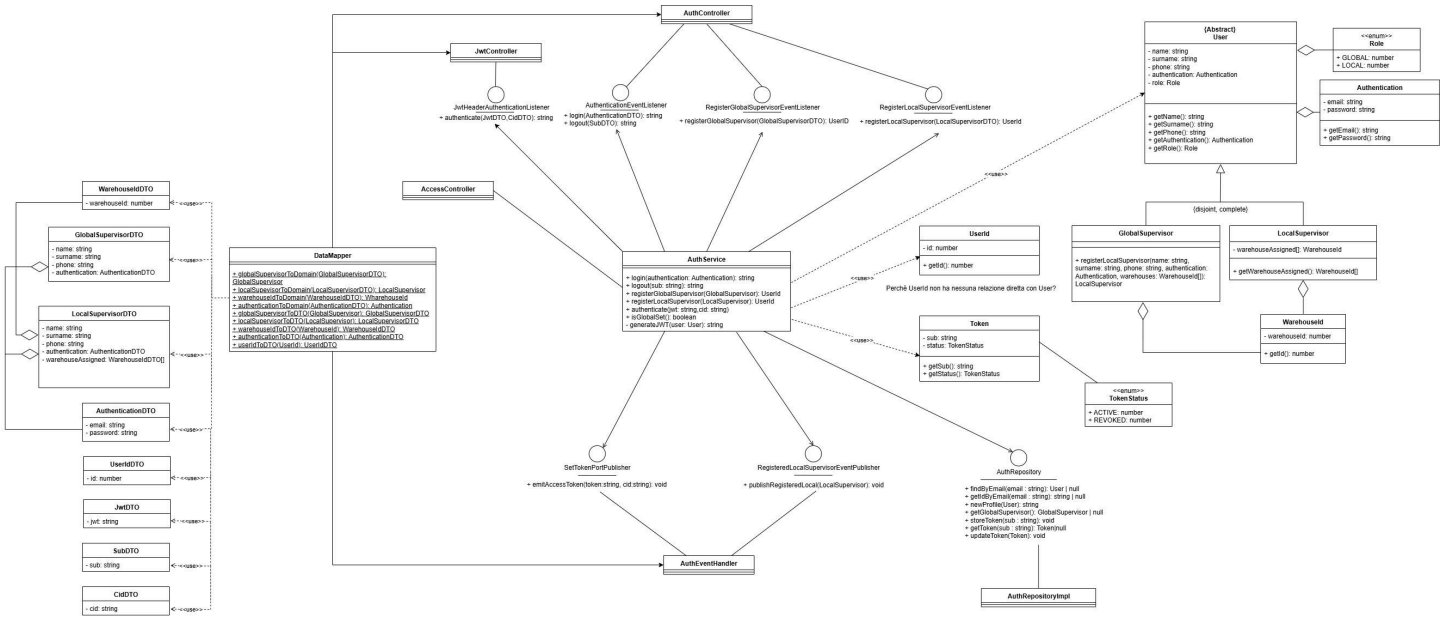

Figura 8: Schema UML - Microservizio Autenticazione

3.5.2.1. Descrizione del microservizio

Il microservizio di Autenticazione si occupa della gestione degli utenti del sistema e della loro autenticazione.

3.5.2.1.1. Funzionalità principali

- **Determinazione delle distanze**: calcolo della distanza tra i magazzini, sulla base delle informazioni disponibili.

3.5.2.2. WarehouseId

1. Rappresenta l'identificatore univoco del magazzino,
2. Incapsula il campo *warehouseId: number*,
3. Espone il metodo *getId()*,
4. È stato isolato per facilitare il confronto tra entità e mantenere l'identità coerente anche in fase di serializzazione/deserializzazione.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  È l'identificativo numerico del magazzino.

Può invocare le seguenti funzioni:

- **getId()**: number
  Metodo pubblico per ottenere l'id del magazzino.

3.5.2.3. `<<enum>>` role

1. Enumerazione che rappresenta il ruolo dei supervisori del sistema.
2. I ruoli possono essere:
   - **GLOBAL**: number
     Rappresenta il ruolo di Supervisore Globale.
   - **LOCAL**: number
     Rappresenta il ruolo di Supervisore Locale.

3.5.2.4. Authentication

1. Rappresenta le credenziali di un utente del sistema.

Descrizione degli attributi della struttura:

- **email**: string
  Rappresenta l'email dell'utente.
- **password**: string
  Rappresenta la password dell'utente.

Può invocare le seguenti funzioni:

- **getEmail()**: string
  Restituisce l'email dell'utente.
- **getPassword()**: string
  Restituisce la password dell'utente.

3.5.2.5. {abstract} User

1. Rappresenta i dati comuni degli utenti del sistema
2. Contiene informazioni personali dell'utente, le sue credenziali e il suo ruolo
3. Uno *User* concreto può essere solo o un *Global Supervisor* o un *Local Supervisor*

Descrizione degli attributi della struttura:

- **name**: string
  Rappresenta il nome dell'utente.
- **surname**: string
  Rappresenta il cognome dell'utente.
- **phone**: string
  Rappresenta il numero di telefono dell'utente.
- **authentication**: Authentication
  Rappresenta le credenziali dell'utente.
- **role**: Role
  Rappresenta il ruolo dell'utente.

Può invocare le seguenti funzioni:

- **getName()**: string
  Restituisce il nome dell'utente.
- **getSurname()**: string
  Restituisce il cognome dell'utente.
- **getPhone()**: string
  Restituisce il numero di telefono dell'utente.
- **getAuthentication()**: Authentication
  Restituisce le credenziali dell'utente.
- **getRole()**: Role
  Restituisce il ruolo dell'utente.

3.5.2.6. Global Supervisor

1. Derivazione di *User*.
2. Rappresenta un Supervisore Globale del sistema.
3. Permette di registrare nuovi Supervisori Locali nel sistema.

Può invocare le seguenti funzioni:

- **registerLocalSupervisor(name: string, surname: string, phone: string, authentication: Authentication, warehouses: WarehouseId[])**: LocalSupervisor
  Registra un nuovo Supervisore Locale nel sistema.

3.5.2.7. Local Supervisor

1. Derivazione di *User*.
2. Rappresenta un Supervisore Locale del sistema.
3. Contiene la lista di magazzini assegnati a quel Supervisore Locale

Descrizione degli attributi della struttura:

- **warehouseAssigned[]**: WarehouseId
  Rappresenta la lista di magazzini assegnati al Supervisore Locale.

Può invocare le seguenti funzioni:

- **getWarehouseAssigned()**: WarehouseId[] Restituisce la lista di magazzini assegnati al Supervisore Locale.

3.5.2.8. UserId

1. Rappresenta il codice identificativo di un utente del sistema.

Descrizione degli attributi della struttura:

- **id**: number
  Rappresenta il codice identificativo di un utente.

Può invocare le seguenti funzioni:

- **getId**: number
  Restituisce il codice identificativo di un utente.

3.5.2.9. `<<enum>>` TokenStatus

1. Enumerazione che rappresenta lo stato di un token utilizzato per l'autenticazione.
2. Gli stati del token possono essere:
   - **ACTIVE**: number
     Rappresenta un token valido.
   - **REVOKED**: number
     Rappresenta un token non valido.

3.5.2.10. Token

1. Rappresenta il token da restituire ai microservizi per la validazione dell'autenticazione dell'utente.

Descrizione degli attributi della struttura:

- **sub**: string
  Rappresenta il subject del token, ossia l'utente identificato.
- **status**: TokenStatus
  Rappresenta lo stato del token.

Può invocare le seguenti funzioni:

- **getSub()**: string
  Restituisce il subject del token.
- **getStatus()**: TokenStatus
  Restituisce lo stato del token.

3.5.2.11. AuthService

1. Rappresenta la logica di business del microservizio di autenticazione.
2. Si occupa di:
   - Registrazione, degli utenti.
   - Autenticazione degli utenti.

Può invocare le seguenti funzioni:

- **login(authentication: Authentication)**: string
  Si occupa dell'autenticazione dell'utente date le sue credenziali.
- **logout(sub: string)**: string
  Si occupa di gestire la fine della sessione dell'utente.
- **registerGlobalSupervisor(GlobalSupervisor)**: UserId
  Registra un nuovo Supervisore Globale a sistema.
- **registerLocalSupervisor(LocalSupervisor)**: UserId
  Registra un nuovo Supervisore Locale a sistema.
- **authenticate(jwt: string,cid: string)**: string
  Si occupa di gestire l'autenticazione mediante il token JWT e il CID.
- **isGlobalSet()**: boolunivoco
  Verifica se esiste già un Supervisore Globale registrato a sistema.
- **generateJWT(user: User)**: string
  Genera il token JWT per l'utente.

3.5.2.12. WarehouseIdDTO

1. Rappresenta il DTO relativo al codice identificativo del magazzino.

Descrizione degli attributi della struttura:

- **warehouseId**: number

3.5.2.13. UserIdDTO

1. Rappresenta il DTO relativo al codice identificativo dell'utente.

Descrizione degli attributi della struttura:

- **id**: number

3.5.2.14. JwtDTO

1. Rappresenta il DTO relativo al token JWT usato per l'autenticazione.

Descrizione degli attributi della struttura:

- **jwt**: string

3.5.2.15. SubDTO

1. Rappresenta il DTO relativo al subject dell'autenticazione.

Descrizione degli attributi della struttura:

- **sub**: string

3.5.2.16. CidDTO

1. Rappresenta il DTO relativo al CID utilizzato per l'autenticazione.

Descrizione degli attributi della struttura:

- **cid**: string

3.5.2.17. AuthenticationDTO

1. Rappresenta il DTO relativo alle credenziali dell'utente.

Descrizione degli attributi della struttura:
- **email**: string
- **password**: string

3.5.2.18. GlobalSupervisorDTO

1. Rappresenta il DTO relativo al Supervisore Globale

Descrizione degli attributi della struttura:
- **name**: string
- **surname**: string
- **phone**: string
- **authentication**: AuthenticationDTO

3.5.2.19. LocalSupervisorDTO

1. Rappresenta il DTO relativo al Supervisore Locale

Descrizione degli attributi della struttura:
- **name**: string
- **surname**: string
- **phone**: string
- **authentication**: AuthenticationDTO
- **warehouseAssigned**: WarehouseIdDTO[]

3.5.2.20. DataMapper

1. Rappresenta la classe in cui sono definiti i metodi per la conversione da oggetti di dominio a DTO e viceversa.

Può invocare le seguenti funzioni:
- **globalSupervisorToDomain(GlobalSupervisorDTO)**: GlobalSupervisor
  Converte un DTO relativo ad un *GlobalSupervisor* al rispettivo oggetto di dominio.
- **localSupevisorToDomain(LocalSupervisorDTO)**: LocalSupervisor
  Converte un DTO relativo ad un *LocalSupervisor* al rispettivo oggetto di dominio.
- **warehouseIdToDomain(WarehouseIdDTO)**: WharehouseId
  Converte un DTO relativo ad un *WarehouseId* al rispettivo oggetto di dominio.
- **authenticationToDomain(AuthenticationDTO)**: Authentication
  Converte un DTO relativo ad un *Authentication* al rispettivo oggetto di dominio.
- **globalSupervisorToDTO(GlobalSupervisor)**: GlobalSupervisorDTO
  Converte un oggetto di dominio di tipo *GlobalSupervisor* al rispettivo DTO.
- **localSupervisorToDTO(LocalSupervisor)**: LocalSupervisorDTO
  Converte un oggetto di dominio di tipo *LocalSupervisor* al rispettivo DTO.
- **warehouseIdToDTO(WarehouseId)**: WarehouseIdDTO
  Converte un oggetto di dominio di tipo *WarehouseId* al rispettivo DTO.
- **authenticationToDTO(Authentication)**: AuthenticationDTO
  Converte un oggetto di dominio di tipo *Authentication* al rispettivo DTO.
- **userIdToDTO(UserId)**: UserIdDTO
  Converte un oggetto di dominio di tipo *UserId* al rispettivo DTO.

3.5.2.21. JwtHeaderAuthenticationListener

1. Porta in entrata per l'autenticazione tramite token JWT e CID.

Può invocare le seguenti funzioni:

- **authenticate(JwtDTO,CidDTO)**: string
  Delega al service l'autenticazione.

3.5.2.22. AuthenticationEventListener

1. Porta in entrata per l'ascolto di eventi di login e logout dell'utente.

Può invocare le seguenti funzioni:

- **login(AuthenticationDTO)**: string
  Delega al service il login dell'utente.
- **logout(SubDTO)**: string
  Delega al service il logut dell'utente.

3.5.2.23. RegisterGlobalSupervisorEventListener

1. Porta in entrata per l'ascolto di comandi di registrazione di un nuovo Supervisore Globale.

Può invocare le seguenti funzioni:

- **registerGlobalSupervisor(GlobalSupervisorDTO)**: UserID

3.5.2.24. RegisterLocalSupervisorEventListener

1. Porta in entrata per l'ascolto di comandi di registrazione di un nuovo Supervisore Locale.

Può invocare le seguenti funzioni:

- **registerLocalSupervisor(LocalSupervisorDTO)**: UserID

3.5.2.25. AuthController

1. Adapter in entrata che implementa le seguenti interfacce:

- **JwtHeaderAuthenticationListener**
- **AuthenticationEventListener**
- **RegisterGlobalSupervisorEventListener**
- **RegisterLocalSupervisorEventListener**

3.5.2.26. SetTokenPortPublisher

1. Porta in entrata per l'invio di token d'accesso.

Può invocare le seguenti funzioni:

- **emitAccessToken(token:string, cid:string)**: void Invia il il token.

3.5.2.27. RegisteredLocalSupervisorEventPublisher

1. Porta in entrata per comunicare l'avvenuta registrazione di un nuovo Supervisore Locale.

Può invocare le seguenti funzioni:

- **publishRegisteredLocal(LocalSupervisor)**: void

3.5.2.28. AuthEventHandler

1. Adapter in uscita che implementa le seguenti interfacce:

- **SetTokenPortPublisher**
- **RegisteredLocalSupervisorEventPublisher**

3.5.3. Microservizio Routing

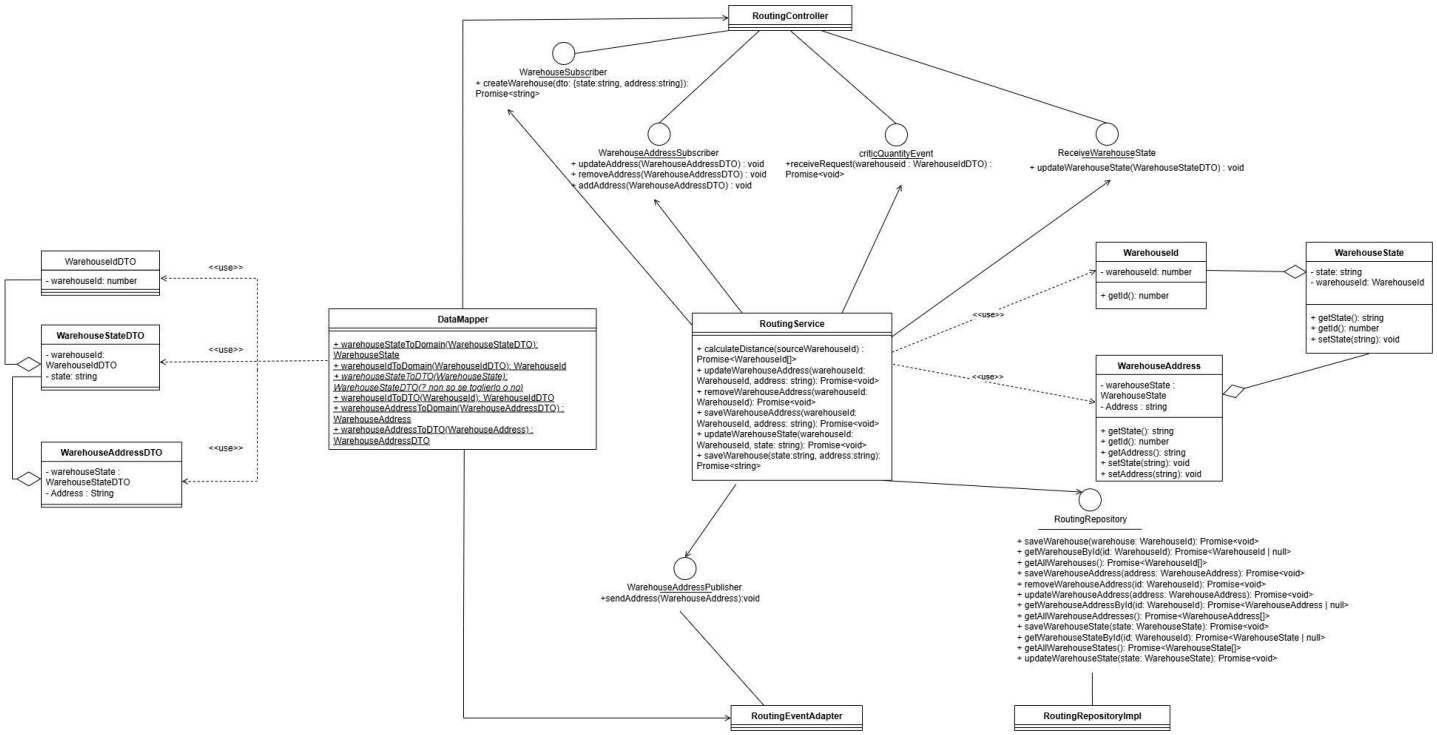

Figura 9: Schema UML - Microservizio Routing

3.5.3.1. Descrizione del microservizio

Il microservizio di Routing è il componente dedicato alla logica di calcolo della distanza e della sequenza ottimale dei magazzini. La sua funzione principale è fornire al Sistema Centralizzato le informazioni necessarie per prendere decisioni rapide ed efficaci in merito bilanciamento delle scorte o gestione degli ordini critici. Questo servizio opera in modo indipendente, concentrandosi unicamente sulle funzionalità di routing, e comunica con gli altri microservizi per ottenere i dati necessari.

3.5.3.1.1. Funzionalità principali

- **Determinazione delle distanze**: calcolo della distanza tra i magazzini, sulla base delle informazioni disponibili.

3.5.3.2. RoutingController

1. Rappresenta il controller incaricato della gestione degli indirizzi di magazzino e della ricezione di eventi legati allo stato dei magazzini.

Può invocare le seguenti funzioni:

- **updateAddress(warehouseAddress: WarehouseAddressDTO)**: void
  Aggiorna un indirizzo di magazzino già registrato nel sistema.
- **removeAddress(warehouseAddress: WarehouseAddressDTO)**: void
  Rimuove un indirizzo di magazzino non più attivo.
- **addAddress(warehouseAddress: WarehouseAddressDTO)**: void
  Aggiunge un nuovo indirizzo di magazzino al sistema di routing.
- **createWarehouse(dto: {state:string, address:string})**: Promise`<string>`
  Crea un nuovo magazzino nel sistema di routing.
- **receiveRequest(warehouseid : WarehouseIdDTO)** : Promise`<void>`
  Riceve la richiesta di calcolo delle distanze da un magazzino critico.

- **updateWarehouseState(warehouseState: WarehouseStateDTO)**: Promise<void>
  Riceve l'aggiornamento dello stato di un magazzino.

3.5.3.3. criticQuantityEvent

1. Rappresenta l'evento emesso dal sistema centralizzato quando si verifica una criticità di quantità. Viene ascoltato dal microservizio **Routing** e viene suddiviso due casi:
   - quando uno o più prodotti dell'inventario di un magazzino scendono sotto la soglia minima,
   - quando un ordine effettuato in un magazzino richiede prodotti e risultano insufficienti.
2. In entrambi i casi, il sistema centralizzato emette questo evento e il microservizio Routing lo riceve per avviare le azioni conseguenti.

Può invocare le seguenti funzioni:

- **receiveRequest(warehouseId: WarehouseIdDTO)**: Promise<void>
  Riceve dal sistema centralizzato l'ID di un magazzino critico e lo inoltra al **RoutingService** per calcolare la sequenza di distanza rispetto agli altri magazzini.

3.5.3.4. ReceiveWarehouseState

1. Rappresenta il componente che gestisce l'aggiornamento dello stato di un magazzino in base ai dati ricevuti.

Può invocare le seguenti funzioni:

- **updateWarehouseState(warehouseState: WarehouseStateDTO)**: Promise<void>
  Aggiorna lo stato di un magazzino in base ai dati ricevuti.

3.5.3.5. WarehouseSubscriber

1. Rappresenta il componente che si occupa di ricevere gli eventi relativi alla creazione di un nuovo magazzino.

Può invocare le seguenti funzioni:

- **createWarehouse(dto: {state:string, address:string})**: Promise<string>
  Crea un nuovo magazzino nel sistema di routing.

3.5.3.6. WarehouseAddressSubscriber

1. Rappresenta il componente che si occupa di ricevere gli eventi relativi alla gestione degli indirizzi dei magazzini.

Può invocare le seguenti funzioni:

- **updateAddress(warehouseAddress: WarehouseAddressDTO)**: void
  Aggiorna un indirizzo di magazzino già registrato nel sistema.
- **removeAddress(warehouseAddress: WarehouseAddressDTO)**: void
  Rimuove un indirizzo di magazzino non più attivo.
- **addAddress(warehouseAddress: WarehouseAddressDTO)**: void
  Aggiunge un nuovo indirizzo di magazzino al sistema di routing.

3.5.3.7. RoutingService

1. Rappresenta il servizio applicativo che si occupa della logica di business relativa al calcolo delle distanze tra magazzini e alla gestione degli indirizzi.

Può invocare le seguenti funzioni:

- **calculateDistance()**: WarehouseId[]
  Esegue il calcolo della distanza tra il magazzino di riferimento e tutti gli altri, restituendo un array di WarehouseId ordinato per distanza.
- **updateWarehouseAddress(warehouseId: WarehouseId, address: string)**: Promise<void>
- **removeWarehouseAddress(warehouseId: WarehouseId)**: Promise<void>
- **saveWarehouseAddress(warehouseId: WarehouseId, address: string)**: Promise<void>
- **updateWarehouseState(warehouseId: WarehouseId, state: string)**: Promise<void>
- **saveWarehouse(state:string, address:string)**: Promise<string>

3.5.3.8. RoutingRepository

1. Rappresenta il repository incaricato della persistenza e recupero delle informazioni relative ai magazzini, indirizzi e stati.

Può invocare le seguenti funzioni:

- **saveWarehouse(warehouse: WarehouseId)**: void
  Salva un nuovo magazzino.
- **getWarehouseById(id: WarehouseId)**: WarehouseId
  Recupera un magazzino tramite il suo ID.
- **getAllWarehouses()**: WarehouseId[]
  Recupera tutti i magazzini.
- **saveWarehouseAddress(address: WarehouseAddress)**: void
  Salva l'indirizzo di un magazzino.
- **removeWarehouseAddress(id: WarehouseId)**: void
  Rimuove l'indirizzo di un magazzino.
- **updateWarehouseAddress(address: WarehouseAddress)**: void
  Aggiorna l'indirizzo di un magazzino.
- **getWarehouseAddressById(id: WarehouseId)**: WarehouseAddress
  Recupera un indirizzo tramite l'ID del magazzino.
- **getAllWarehouseAddresses()**: WarehouseAddress[]
  Recupera tutti gli indirizzi dei magazzini.
- **saveWarehouseState(state: WarehouseState)**: void
  Salva lo stato di un magazzino.
- **getWarehouseStateById(id: WarehouseId)**: WarehouseState
  Recupera lo stato di un magazzino tramite il suo ID.
- **getAllWarehouseStates()**: WarehouseState[]
  Recupera gli stati di tutti i magazzini.
- **updateWarehouseState(state: WarehouseState)**: Promise<void>
  Aggiorna lo stato di un magazzino.

3.5.3.9. DataMapper

1. Rappresenta il componente che si occupa della conversione tra oggetti di dominio e DTO, assicurando la separazione dei livelli applicativi.

Può invocare le seguenti funzioni:

- **toDomain(warehouseState: WarehouseStateDTO)**: WarehouseState
  Converte un DTO WarehouseStateDTO in un oggetto di dominio WarehouseState.
- **toDomain(warehouseId: WarehouseIdDTO)**: WarehouseId
  Converte un DTO WarehouseIdDTO in un oggetto di dominio WarehouseId.

- **toDomain(warehouseAddress: WarehouseAddressDTO)**: WarehouseAddress
  Converte un DTO WarehouseAddressDTO in un oggetto di dominio WarehouseAddress.
- **toDTO(warehouseState: WarehouseState)**: WarehouseStateDTO
  Converte un oggetto di dominio WarehouseState in un DTO WarehouseStateDTO.
- **toDTO(warehouseId: WarehouseId)**: WarehouseIdDTO
  Converte un oggetto di dominio WarehouseId in un DTO WarehouseIdDTO.
- **toDTO(warehouseAddress: WarehouseAddress)**: WarehouseAddressDTO
  Converte un oggetto di dominio WarehouseAddress in un DTO WarehouseAddressDTO.

3.5.3.10. RoutingEventAdapter

1. Rappresenta l'adapter responsabile della pubblicazione degli eventi relativi agli indirizzi, alle distanze dei magazzini e lo stato dei magazzini.

Può invocare le seguenti funzioni:
- **sendAddress(warehouseAddress: WarehouseAddress)**: void
  Invia un evento con i dati di un indirizzo.
- **sendWarehouseDistance(warehouseId: WarehouseId)**: void
  Invia un evento con le distanze calcolate.
- **RequestWarehouseState(WarehouseState)**:void
  Invia una richiesta al microservizio **Cloud State** per essere a conoscenza dello stato di un/o più Warehouse.

3.5.3.11. WarehouseIdDTO

1. DTO per trasferire l'identificativo di un magazzino.

Descrizione degli attributi della struttura:
- **warehouseId**: number
  L'ID del magazzino.

3.5.3.12. WarehouseStateDTO

1. DTO per trasferire lo stato di un magazzino.

Descrizione degli attributi della struttura:
- **warehouseId**: WarehouseIdDTO
  L'ID del magazzino a cui si riferisce lo stato.
- **state**: string
  Lo stato del magazzino.

3.5.3.13. WarehouseAddressDTO

1. DTO per trasferire l'indirizzo di un magazzino.

Descrizione degli attributi della struttura:
- **warehouseState**: WarehouseStateDTO
  Lo stato del magazzino associato.
- **address**: string
  L'indirizzo del magazzino.

3.5.3.14. WarehouseId

1. Rappresenta l'identificativo univoco di un magazzino.

Descrizione degli attributi della struttura:

- **warehouseId**: number
  L'ID numerico del magazzino.

Può invocare le seguenti funzioni:
- **getId()**: number
  Restituisce l'ID del magazzino.

3.5.3.15. WarehouseState

1. Rappresenta lo stato operativo di un magazzino.

Descrizione degli attributi della struttura:
- **state**: string
  Lo stato attuale del magazzino.
- **warehouseId**: WarehouseId
  Il riferimento all'ID del magazzino a cui lo stato appartiene.

Può invocare le seguenti funzioni:
- **getState()**: string
  Restituisce lo stato.
- **getId()**: number
  Restituisce l'ID del magazzino associato.

3.5.3.16. WarehouseAddress

1. Rappresenta l'indirizzo fisico di un magazzino, combinando lo stato e l'indirizzo stesso.

Descrizione degli attributi della struttura:
- **warehouseState**: WarehouseState
  Lo stato del magazzino associato a questo indirizzo.
- **address**: string
  L'indirizzo fisico completo.

Può invocare le seguenti funzioni:
- **getState()**: string
  Restituisce lo stato del magazzino.
- **getId()**: number
  Restituisce l'ID del magazzino.
- **getAddress()**: string
  Restituisce l'indirizzo.

4. Stato dei requisiti funzionali

4.1. Stato per requisito

Verranno ora elencati i requisiti[G] del sistema, che sono stati suddivisi in quattro categorie principali: Requisiti[G] Funzionali, Requisiti[G] di Qualità, Requisiti[G] di Vincolo, Requisiti[G] Prestazionali.

4.2. Classificazione dei requisiti

- **Requisiti[G] Funzionali:** descrivono le funzionalità specifiche che il sistema deve offrire. Definiscono i comportamenti attesi in risposta a determinati input o situazioni, specificando cosa il sistema deve fare per soddisfare i bisogni degli utenti e degli stakeholder.
- **Requisiti[G] di Qualità:** detti anche non funzionali, definiscono le caratteristiche generali del sistema che ne influenzano l'efficacia, l'efficienza e l'affidabilità. Rientrano in questa

categoria aspetti come la sicurezza, l'usabilità, la manutenibilità, la scalabilità[G] e l'affidabilità complessiva del sistema.

- **Requisiti[G] di Vincolo:** specificano le limitazioni imposte da fattori esterni o immutabili, che il sistema o il processo di sviluppo devono rispettare. Tali vincoli possono derivare da normative, tecnologie obbligatorie, standard industriali, vincoli temporali o economici.
- **Requisiti[G] di Prestazionali:** definiscono le aspettative in termini di prestazioni del sistema, come tempi di risposta, capacità di carico, throughput e uso delle risorse. Questi requisiti[G] sono fondamentali per garantire un'esperienza utente adeguata anche sotto carico elevato.

4.3. Fonti dei requisiti

Le fonti dei requisiti[G] rappresentano i documenti e le informazioni da cui sono stati estratti i requisiti[G] stessi. Tra le principali fonti si annoverano il capitolato[G] d'appalto, le riunioni con il committente[G], l'analisi dello stato dell'arte e l'analisi dei casi d'uso.

Ogni requisito[G] riportato sarà accompagnato dall'indicazione esplicita della propria fonte di provenienza, al fine di garantirne la tracciabilità e la verificabilità[G].

4.4. Struttura della codifica dei requisiti

I requisiti[G] sono stati codificati al fine di facilitarne la lettura, la gestione e la tracciabilità. Ogni codice è composto da un prefisso che indica la tipologia del requisito[G], seguito da un numero progressivo univoco.

I requisiti[G] funzionali sono preceduti dal prefisso «RF», i Requisiti[G] di Qualità dal prefisso «RQ», i Requistiti[G] di Vincolo dal prefisso «RV» e i Requisiti[G] Prestazionali dal prefisso «RP», dove:

- **R** sta per «**R**equisito»;
- **F** sta per «**F**unzionale»;
- **Q** sta per «**Q**ualità»;
- **V** sta per «**V**incolo»;
- **P** sta per «**P**restazionale»;

Per facilitare la lettura, la tracciabilità e la classificazione dei requisiti[G], è stato adottato un sistema di codifica sturtturato. La codifica prevede un prefisso che identifica la tipologia e l'importanza del requisito[G], seguito da un numero progressivo. In caso di scomposizione, si aggiunge una notazione per indicare i requisiti derivati.

4.4.1. Tipologia e Importanza

I requisiti[G] sono stati distinti anche in base alla loro importanza o natura, secondo le seguenti convenzioni:

- **Standard**: requisiti[G] strettamente necessari al corretto funzionamento del sistema.
  - *Esempio:* RF01 → Requisito Funzionale 01.
- **Desiderabili (D)**: requisiti[G] non obbligatori, ma in grado di apportare un valore aggiunto al sistema.
  - *Esempio:* RFD04 → Requisito Funzionale Desiderabile 02.
- **Opzionali (O)**: requisiti[G] implementabili solo in presenza di tempo o risorse sufficienti.
  - *Esempio:* RFO03 → Requisito Funzionale Opzionale 02.

4.5. Requisiti Funzionali

Alcuni requisiti funzionali legati all'interfaccia grafica non sono stati implementati, poiché l'azienda committente ha esplicitamente espresso la volontà di privilegiare lo sviluppo della logica di sistema e dell'architettura software. Tali requisiti sono pertanto classificati come non soddisfatti per motivi di priorità progettuale.

Codice	Descrizione	Stato
RF01	**Registrazione del Supervisore Globale.**	
RF01/01	Il Supervisore Globale deve registrarsi al primo avvio del sistema inserendo indirizzo email, numero di cellulare e una password.	Soddisfatto
RF01/02	Il Supervisore Globale visualizza, in fase di registrazione, un messaggio di errore se l'email inserita è sintatticamente errata.	Soddisfatto
RF01/03	Il Supervisore Globale visualizza, in fase di registrazione, un messaggio di errore se il numero di cellulare inserito è sintatticamente errato.	Soddisfatto
RF01/04	Il Supervisore Globale visualizza, in fase di registrazione, un messaggio di errore se la password inserita non rispetta la sintassi.	Soddisfatto
RF01/05	Il Supervisore Globale visualizza, in fase di registrazione, un messaggio di errore se la password e la conferma password non corrispondono.	Soddisfatto
RF02	**Autenticazione[G] dell'utente.**	
RF02/01	L'utente deve poter accedere al sistema mediante credenziali (email e password).	Soddisfatto
RF02/02	L'utente deve poter effettuare il logout dal sistema.	Soddisfatto
RF02/03	L'utente visualizza un messaggio di errore in caso di autenticazione[G] fallita.	Soddisfatto
RF03	**Registrazione di nuovi Supervisori Locali.**	
RF03/01	Il Supervisore Globale può registrare a sistema nuovi Supervisori Locali ciascuno con email, numero di cellulare e una password.	Soddisfatto

Codice	Descrizione	Stato
RF03/02	Il Supervisore Globale deve assegnare il/i magazzino/i a cui il Supervisore Locale ha accesso in fase di registrazione.	Soddisfatto
RF03/03	Il Supervisore Globale visualizza, in fase di registrazione del nuovo supervisore locale, un messaggio di errore se l'email inserita è sintatticamente errata.	Non Soddisfatto
RF03/04	Il Supervisore Globale visualizza, in fase di registrazione del nuovo supervisore locale, un messaggio di errore se la password inserita non rispetta la sintassi.	Non Soddisfatto
RF03/05	Il Supervisore Globale visualizza, in fase di registrazione del nuovo supervisore locale, un messaggio di errore se il numero di cellulare inserito è sintatticamente errato.	Non Soddisfatto
RF04	**Gestione manuale dei magazzini.**	
RF04/01	Il Supervisore Globale deve poter aggiungere un nuovo magazzino a sistema inserendo il suo indirizzo fisico.	Soddisfatto
RF04/02	Il Supervisore Globale deve poter rimuovere un magazzino dal sistema.	Soddisfatto
RF04/03	Il Supervisore Globale deve poter modificare le informazioni di un magazzino, quali indirizzo del magazzino e/o Supervisore Locale associato.	Soddisfatto
RF04/04	Ciascun magazzino deve essere identificato univocamente.	Soddisfatto
RF05	**Gestione manuale delle merci.**	
RF05/01	I Supervisori possono inserire un nuovo tipo di merce nell'inventario[G] di un magazzino, ciascuna con codice univoco e nome prodotto[G].	Soddisfatto
RF05/02	I Supervisori visualizzano un messaggio di errore se il codice univoco non rispetta la sintassi prevista.	Non Soddisfatto
RF05/03	I Supervisori possono definire il prezzo unitario all'inserimento di un nuovo tipo di merce nell'inventario[G] di un magazzino.	Soddisfatto

Codice	Descrizione	Stato
RF05/04	I Supervisori visualizzano un messaggio di errore se il prezzo unitario del prodotto[G] è sintatticamente errato.	Soddisfatto
RF05/05	I Supervisori possono definire la quantità all'inserimento di un nuovo tipo di merce nell'inventario[G] di un magazzino.	Soddisfatto
RF05/06	I Supervisori visualizzano un messaggio di errore se la quantità del prodotto[G] è sintatticamente errata.	Non Soddisfatto
RF05/07	Il Supervisore Globale può rimuovere un tipo di merce dall'inventario[G].	Soddisfatto
RF05/08	I Supervisori possono modificare la quantità di merce nell'inventario[G] dei magazzini.	Soddisfatto
RF05/09	I Supervisori visualizzano un messaggio di errore se la quantità di merce modificata del prodotto[G] è sintatticamente errata.	Non Soddisfatto
RF05/10	Il Supervisore Globale può modificare il prezzo unitario di un prodotto[G].	Soddisfatto
RF06	**Gestione manuale degli ordini[G].**	
RF06/01	I Supervisori possono inserire ordini[G] di trasferimento[G] interno tra magazzini selezionando il magazzino di partenza e inserendo uno o più prodotti[G] nell'ordine[G].	Soddisfatto
RF06/02	I Supervisori possono inserire ordini[G] di vendita verso l'esterno inserendo l'indirizzo del destinatario e inserendo uno o più prodotto[G] nell'ordine[G].	Soddisfatto
RF06/03	I Supervisori visualizzano un messaggio di errore se la quantità di prodotto[G] in un ordine[G] non rispetta la sintassi prevista.	Soddisfatto
RF06/04	I Supervisori possono annullare ordini[G] «in attesa» e «in elaborazione».	Soddisfatto
RF07	**Auditing dei dati dei magazzini.**	Soddisfatto
RF07/01	Il Supervisore Globale può visualizzare l'inventario[G] globale.	Soddisfatto

Codice	Descrizione	Stato
RF07/02	Il Supervisore Globale può visualizzare l'inventario[G] di ciascun magazzino.	Soddisfatto
RF07/03	Il Supervisore Globale può visualizzare un report degli ordini[G] globali.	Soddisfatto
RF07/04	Il Supervisore Globale può visualizzare un report degli ordini[G] di ciascun magazzino.	Soddisfatto
RF07/05	I Supervisori Locali possono visualizzare l'inventario[G] del/dei magazzino/i a loro assegnato/i.	Soddisfatto
RF07/06	I Supervisori Locali possono visualizzare un report degli ordini[G] del/dei magazzino/i a loro assegnato/i.	Soddisfatto
RF08	I Supervisori Globali possono visualizzare lo stato di tutti i magazzini.	Soddisfatto
RF09	**Gestione delle richieste.**	
RF09/01	I Supervisori possono visualizzare le notifiche informative prodotte dal sistema.	Soddisfatto
RF09/02	Il Supervisore Globale può accettare le richieste decisionali prodotte dal sistema.	Non Soddisfatto
RF09/03	Il Supervisore Globale può rifiutare le richieste decisionali prodotte dal sistema.	Non Soddisfatto
RF10	**Gestione delle soglie critiche.**	
RF10/01	I Supervisori possono definire i valori di soglia minima all'inserimento di nuovo tipo di merce nell'inventario[G] di un magazzino.	Soddisfatto
RF10/02	I Supervisori possono definire i valori di soglia massima all'inserimento di nuovo tipo di merce nell'inventario[G] di un magazzino.	Soddisfatto
RF10/03	I Supervisori visualizzano un messaggio di errore se i valori di soglia minima e/o massima inseriti sono sintatticamente errati.	Soddisfatto
RF10/04	Il Supervisore Globale può modificare i valori di soglia minima e massima di ciascun tipo di merce per ciascun magazzino.	Soddisfatto

Codice	Descrizione	Stato
RF10/05	Il Supervisore Globale visualizza un messaggio di errore se i valori di soglia minima e/o massima modificati sono sintatticamente errati.	Non Soddisfatto
RF10/06	I Supervisori Locali possono modificare i valori di soglia minima e massima di ciascun tipo di merce nel/nei magazzino/i a loro assegnato/i.	Soddisfatto
RF10/07	I Supervisori Locali visualizzano un messaggio di errore se i valori di soglia minima e/o massima modificati sono sintatticamente errati.	Non Soddisfatto
RF11	**Rilevamento di carenza di scorte[G] di un magazzino.**	
RF11/01	Ciascun magazzino deve identificare quando le scorte[G] scendono sotto una certa soglia.	Soddisfatto
RF11/02	Ciascun magazzino deve identificare quando le scorte[G] salgono sopra una certa soglia.	Soddisfatto
RF12	**Gestione degli ordini[G].**	
RF12/01	Ciascun magazzino deve verificare[G] la disponibilità delle scorte[G] per poter soddisfare un ordine[G] ricevuto.	Soddisfatto
RF12/02	Ciascun magazzino deve identificare l'impossibilità di soddisfare un ordine[G].	Soddisfatto
RF12/03	Un magazzino con scorte[G] insufficienti per gestire un ordine[G], deve richidere un riassortimento[G] al sistema.	Soddisfatto
RF12/04	Ciascun magazzino deve tracciare lo stato degli ordini[G].	Soddisfatto
RF12/05	I magazzini possono inviare merce verso un altro magazzino tramite un ordine[G].	Soddisfatto
RF12/06	I magazzini possono ricevere approvvigionamenti[G] dall'esterno tramite ordine[G].	Soddisfatto
RF12/07	I magazzini possono spedire merci verso l'esterno tramite ordine[G].	Soddisfatto
RF12/08	Il magazzino di destinazione deve notificare l'arrivo della merce al magazzino di origine.	Soddisfatto

Codice	Descrizione	Stato
RF12/09	Un ordine[G] deve trovarsi in uno dei seguenti stati: «in attesa», «in elaborazione», «in transito», «annullato», «consegnato».	Soddisfatto
RF13	**Riassortimento[G] delle scorte[G] tra magazzini.**	
RF13/01	Il sistema deve programmare trasferimenti interni al raggiungimento di una soglia critica di un prodotto[G] in un magazzino al fine di bilanciare le scorte[G].	Soddisfatto
RF14	**Individuazione dello stato dei magazzini.**	
RF14/01	Ciascun magazzino può trovarsi in uno dei seguenti stati: «online», «offline».	Soddisfatto
RF14/02	Il sistema deve identificare eventuali cambi di stato dei magazzini.	Soddisfatto
RF14/03	Il sistema deve identificare eventuali disconnessioni di uno o più magazzini segnandoli come «offline».	Soddisfatto
RF15	**Gestione dei disservizi.**	
RF15/01	Un magazzino offline deve annullare gli ordini[G] che non può soddisfare dopo un periodo di tempo predefinito.	Soddisfatto
RF15/02	Il Supervisore Globale può definire il periodo di tempo dopo cui l'ordine[G] di un magazzino offline viene automaticamente anullato.	Soddisfatto
RF16	**Gestione magazzino che cambia stato da offline a online.**	
RF16/01	Il sistema deve riconoscere quando un magazzino torna online.	Soddisfatto
RF17	**Annullamento di ordini[G] in transito.**	Soddisfatto
RF17/01	Un ordine[G] in stato «in transito» deve cambiare lo stato in «annullato» dopo una soglia temporale predefinita.	Soddisfatto
RF17/02	Il Supervisore Globale può definire la soglia temporale dopo cui un ordine[G] «in transito» cambia automaticamente stato in «annullato».	Non Soddisfatto

Codice	Descrizione	Stato
RFD01	Il Supervisore Globale può definire il costo massimo per i trasferimenti eseguibili mediante riassortimento[G] automatico.	Non Soddisfatto
RFD02	Il Supervisore Globale può definire la distanza massima tra magazzini per il riassortimento[G] automatico.	Soddisfatto
RFD03	**Suggerimento su azioni di riassortimento[G] e trasferimento[G] tra magazzini.**	
RFD03/01	Il sistema invia una richiesta decisionale al Supervisore Globale qualora un riassortimento[G] superi i costi o le distanze massime.	Non Soddisfatto
RFD04	**Interfaccia grafica.**	
RFD04/01	Deve essere possibile effettuare la registrazione del Supervisore Globale mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/02	Deve essere possibile effettuare l'autenticazione[G] dell'utente mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/03	Deve essere possibile effettuare la registrazione di nuovi Supervisori Locali mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/04	Deve essere possibile effettuare la gestione manuale delle merci mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/05	Deve essere possibile effettuare la gestione manuale degli ordini[G] mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/06	Deve essere possibile effettuare l'auditing dei dati dei magazzini mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/07	Deve essere possibile effettuare la gestione delle richieste mediante Interfaccia Grafica.	Non Soddisfatto
RFD04/08	Deve essere possibile effettuare la gestione delle soglie critiche mediante Interfaccia Grafica.	Non Soddisfatto
RFD05	**Produzione di notifiche.**	
RFD05/01	Il Sistema deve notificare i Supervisori al superamento dei livelli minimi e massimi di scorte[G].	Non Soddisfatto

Codice	Descrizione	Stato
RFD05/02	Il Sistema deve notificare i Supervisori dell'esecuzione del riassortimento[G] automatico.	Non Soddisfatto
RFD05/03	Ciascun magazzino deve notificare i Supervisori dei cambi di stato degli ordini[G].	Non Soddisfatto
RFD05/04	Ciascun magazzino deve notificare i Supervisori dell'impossibilità di soddisfare un ordine[G].	Non Soddisfatto
RFD05/05	Ciascun magazzino deve notificare i Supervisori quando le scorte[G] scendono sotto una certa soglia.	Non Soddisfatto
RFD05/06	Ciascun magazzino deve notificare i Supervisori quando le scorte[G] salgono sopra una certa soglia.	Non Soddisfatto
RFD05/07	Il sistema deve notificare il Supervisore Globale di eventuali cambi di stato dei magazzini.	Non Soddisfatto
RFD05/08	L'invio di notifiche deve poter avvenire tramite email.	Non Soddisfatto
RFD05/09	L'invio di notifiche deve poter avvenire tramite SMS.	Non Soddisfatto
RFD06	**Produzione di richieste decisionali.**	
RFD06/01	Il Sistema deve inviare al Supervisore Globale le richieste decisionali prodotte dal riassortimento[G] predittivo.	Non Soddisfatto
RFD07	**Riassortimento predittivo.**	
RFD07/01	Il sistema deve avere uno storico dei livelli di merce.	Non Soddisfatto
RFD07/02	Il sistema deve poter analizzare i dati storici degli ordini[G].	Non Soddisfatto
RFD07/03	Il sistema deve poter analizzare i dati storici di inventario[G] dei magazzini.	Non Soddisfatto
RFD07/04	Il sistema deve implementare un modello di previsione della domanda dei prodotti[G].	Non Soddisfatto
RFD07/05	Il sistema deve poter pianifiare i riassortimenti[G] sulla base del modello di previsione.	Non Soddisfatto

Codice	Descrizione	Stato
RFD07/06	Il sistema deve inviare una richiesta decisionale al Supervisore Globale di approvvigionamento[G] sulla base del modello di previsione.	Non Soddisfatto
RFD07/07	Il sistema deve inviare una richiesta decisionale al Supervisore Globale di riassortimento[G] sulla base del modello di previsione.	Non Soddisfatto
RFO01	I supervisori devono poter eseguire un ripristino[G] manuale dei dati da un backup[G].	Non Soddisfatto
RFO02	**Gestione dei magazzini non operativi.**	
RFO02/01	Ciascun magazzino può trovarsi nello stato «non operativo».	Non Soddisfatto
RFO02/02	Il sistema può annullare ordini[G] di un magazzino «non operativo».	Soddisfatto
RFO02/03	Il sistema può assegnare ordini[G] di un magazzino «non operativo» ad altri magazzini.	Non Soddisfatto

Tabella 8: Requisiti Funzionali

4.6. Requisiti di Vincolo

Codice	Descrizione	Stato
RV01	L'architettura[G] del sistema deve usare microservizi[G].	Soddisfatto
RV02	Versionamento del codice tramite Git.	Soddisfatto

Tabella 9: Requisiti di Vincolo

4.7. Requisiti di Qualità

Codice	Descrizione	Stato
RQ01	**Autonomia dei magazzini.**	
RQ01/01	I magazzini devono poter operare autonomamente.	Soddisfatto
RQ01/02	I magazzini devono poter elaborare localmente le richieste.	Soddisfatto
RQ01/03	Ciascun magazzino deve gestire gli ordini[G] concorrenti per uno stesso prodotto secondo una politica predefinita.	Soddisfatto

Codice	Descrizione	Stato
RQ02	**Robustezza dei magazzini.**	
RQ02/01	I dati di inventario[G] e degli ordini[G] devono essere coerenti, consistenti e persistenti in tutto il sistema.	Soddisfatto
RQ02/02	I dati di inventario[G] devono essere resilienti a modifiche concorrenti.	Soddisfatto
RQ03	**Sicurezza del sistema.**	
RQ03/01	Deve essere implementata la cifratura[G] end-to-end negli scambi dei dati tra magazzini e con il sistema.	Non Soddisfatto
RQ03/02	Il sistema deve implementare un log dei tentativi di accesso.	Soddisfatto
RQ03/03	Il sistema deve identificare tentativi di accesso non autorizzati.	Soddisfatto
RQ03/04	Il sistema deve notificare il Supervisore Globale di tentativi di accesso non autorizzati.	Non Soddisfatto
RQ04	**Unit test e code coverage.**	
RQ04/01	La correttezza delle funzioni sviluppate deve essere garantita dai test unitari.	Soddisfatto
RQ04/02	La copertura del codice deve essere superiore al 70% (minimo) e 80% (auspicabile).	Soddisfatto (83%)
RQ04/03	Il test devono poter essere eseguiti in maniera automatizzata.	Soddisfatto
RQ04/04	Devono essere eseguiti test di non regressione.	Soddisfatto
RQ05	**Test book dettagliato.**	
RQ05/01	Tutti i test devono essere documentati in un test book.	Soddisfatto
RQ05/02	Il test book deve includere una descrizione dei casi di test.	Soddisfatto
RQ05/03	Il test book deve includere le condizioni iniziali dei test.	Soddisfatto
RQ05/04	Il test book deve includere i passaggi eseguiti dai test.	Soddisfatto

Codice	Descrizione	Stato
RQ05/05	Il test book deve includere i risultati attesi e ottenuti dai test.	Soddisfatto
RQ05/06	Il test book deve includere criteri di validazione[G] dei test.	Soddisfatto
RQ06	Il Sistema deve essere scalabile orizzontalmente.	Soddisfatto
RQ07	La gestione dei magazzini deve essere distribuita.	Soddisfatto
RQD01	Il sistema deve automatizzare il processo di riassortimento[G] se rispetta il costo e la distanza massima.	Soddisfatto
RQD02	II magazzino detiene la fonte affidabile dei dati di inventario[G] e degli ordini[G].	Soddisfatto
RQO01	**Sicurezza del sistema.**	
RQO01/01	Il sistema deve prevedere la possibilità di autenticazione[G] a due fattori (2FA[G]) o più (MFA[G]) per gli utenti.	Non Soddisfatto
RQO01/02	Il sistema deve verificare[G] l'integrità[G] dei dati tramite firme digitali[G] o hashing[G].	Non Soddisfatto
RQO01/03	Le richieste verso i microservizi[G] devono essere autenticate.	Soddisfatto
RQO01/04	Gli scambi di dati tra microservizi[G] devono essere cifrati.	Non Soddisfatto
RQO02	**Backup[G] e ripristino[G] dei dati.**	
RQO02/01	I dati dei magazzini devono essere sottoposti a backup[G].	Non Soddisfatto
RQO02/02	I dati aggregati su cloud devono essere sottoposti a backup[G].	Non Soddisfatto
RQO02/03	I magazzini devono mantenere una copia locale dei propri dati.	Non Soddisfatto
RQO02/04	Il sistema deve mantenere l'integrità[G] e la consistenza dei dati tra i magazzini.	Soddisfatto

Tabella 10: Requisiti di Qualità