Specifica Tecnica
Versione 1.0.0

Stato	Approvato
Responsabile	Matteo Schievano
Verificatore	Emanuele Artusi Marco Piccoli Matteo Schievano Sara Ferraro
Redattori	Loris Libralato Matteo Schievano Sara Ferraro Samuele Esposito
Distribuzione	*ALimitedGroup* M31 Prof. Tullio Vardanega Prof. Riccardo Cardin

Descrizione

Questo documento contiene la *Specifica Tecnica* descritto dal gruppo *ALimitedGroup* per il Capitolato numero 6 proposto da M31

Registro delle Modifiche

Vers.	Data	Autore	Verificatore	Descrizione
1.0.0	2025-04-04	S. Esposito	L. Libralato	Aggiunta di notifications e api gateway. (Sezione 3.4.11 e Sezione 3.4.12). Approvazione da parte di M. Schievano.
0.9.0	2025-03-31	M. Schievano	M. Piccoli	Aggiunte informazioni sulle versioni delle componenti utilizzate e aggiunte informazioni sul funzionamento di main e router. Aggionte versioni delle librerie. Aggiunte informazioni sulla telemetria. (Sezione 3.4.1, Sezione 3.4.2 e Sezione 3.4.4).
0.8.0	2025-03-27	L. Libralato	M. Piccoli	Descrizione del microservizio order (Sezione 3.4.8).
0.7.0	2025-03-17	M. Schievano	S. Ferraro	Descrizione del microservizio authenticator (Sezione 3.4.7).
0.6.0	2025-03-16	S. Ferraro	M. Schievano	Descrizione pattern Adapter (Sezione 3.3.2).
0.5.0	2025-03-12	S. Ferraro	M. Schievano	Descrizione Pattern Dependency Injection (Sezione 3.3.1).
0.4.0	2025-03-13	L. Libralato	M. Piccoli	Descrizione del microservizio Warehouse (Sezione 3.4.10).
0.3.0	2025-03-09	M. Schievano	M. Piccoli	Descrizione del microservizio Catalog (Sezione 3.4.9).
0.2.0	2025-02-27	S. Ferraro	M. Piccoli	Continuazione sezione architettura (Sezione 3).
0.1.0	2025-02-25	S. Ferraro	E. Artusi	Prima redazione documento. Sezione introduzione. Sezione tecnologie. Sezione architettura (Sezione 1 e Sezione 3).

Indice

Lista delle tabelle

Lista delle immagini

1 - Introduzione

1.1 - Scopo del documento

Il presente documento ha l'obiettivo di descrivere in dettaglio l'*architettura* del prodotto software, fornendo una visione chiara e strutturata delle sue componenti, della loro interazione e della loro distribuzione nel sistema.

Il documento di **Specifica Tecnica**[G] funge da riferimento per la *progettazione* e *realizzazione del prodotto*, garantendo coerenza con il *Proof of Concept*[G] (PoC[G]) iniziale e introducendo miglioramenti volti a consolidarne la maturità architetturale.

Nello specifico, questo documento si propone di:

- Definire l'**architettura logica** del prodotto, illustrando le componenti principali, i loro ruoli e le interconnessioni tra di esse;
- Esporre l'**architettura di deployment**[G], delineando la distribuzione delle componenti nel sistema in esecuzione;
- Documentare i **design pattern architetturali** adottati, evidenziando le scelte progettuali derivate dalle tecnologie selezionate;
- Identificare eventuali **idiomi** (pattern di livello inferiore) utilizzati per ottimizzare la qualità del codice;
- Fornire ulteriori **dettagli progettuali** che valorizzino le scelte architetturali e facilitino la comprensione e manutenzione[G] del prodotto.

1.2 - Glossario

Per tutte le *definizioni*, *acronimi* e *abbreviazioni* utilizzati in questo documento, si faccia riferimento al **Glossario**, fornito come documento separato, che contiene tutte le spiegazioni necessarie per garantire una comprensione uniforme dei termini tecnici e dei concetti rilevanti per il progetto.
Le parole che possiedono un riferimento nel Glossario saranno indicate nel modo che segue:

`parola`[G]

1.3 - Riferimenti

1.3.1 - Riferimenti normativi

- **Capitolato d'appalto C6: Sistema di Gestione di un Magazzino Distribuito - M31**
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C6.pdf
  Ultimo Accesso 4 Aprile 2025
- **Norme di Progetto**[G] **ver. 2.0.0**
  https://alimitedgroup.github.io/NP%20v2.0.0.pdf
  Ultimo Accesso 4 Aprile 2025

1.3.2 - Riferimenti informativi

- **Lezione rovesciata - Documentazione**
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/FC1.pdf
  Ultimo Accesso 4 Aprile 2025

- **Regolamento del Progetto didattico**
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf
  Ultimo Accesso 4 Aprile 2025
- **Glossario**
  https://alimitedgroup.github.io/Glossario.pdf
  Ultimo Accesso 4 Aprile 2025

2 - Tecnologie

Il progetto si basa su un insieme di tecnologie moderne e robuste, selezionate per le loro capacità di supportare efficacemente un'architettura a microservizi e garantire scalabilità, affidabilità e manutenibilità del sistema.

La scelta tecnologica è stata guidata dalla necessità di creare un sistema di gestione del magazzino distribuito che possa operare in modo efficiente anche in condizioni di carico variabile, mantenendo elevati standard di prestazioni e resilienza.

Le tecnologie adottate sono state organizzate 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*, e piattaforme per il monitoraggio del sistema.

Di seguito sono elencate e descritte le tecnologie utilizzate, evidenziando le loro caratteristiche principali.

2.1 - Linguaggi di programmazione

Tecnologia	Versione	Descrizione
GO	1.24.0	Go è un linguaggio di programmazione open-source sviluppato da Google, progettato per essere efficiente, semplice e scalabile. È particolarmente adatto per lo sviluppo di sistemi distribuiti, microservizi e applicazioni cloud-native, grazie alla sua compilazione rapida, alla gestione automatica della memoria e alla facilità di deployment con binari standalone.

Tabella 1: Linguaggi di programmazione

2.2 - Framework per la codifica

Tecnologia	Versione	Descrizione
fx	1.23.0	**Fx** è un *framework* per la *dependency injection* utilizzabile con il linguaggio **Go** sviluppato da *Uber* Per maggiori informazioni si consiglia la consultazione del sito web del progetto.
gin	1.10.0	*Framework* che permette di gestire gli *endpoint* e non solo. Per maggiori informazioni si rimanda alla pagina Go del progetto

Tabella 2: Framework per la codifica

2.3 - Tecnologie per la gestione di dati temporali

Tecnologia	Versione	Descrizione
InfluxDB	2.7.11	Si tratta di una base di dati specificatamente progettata per la memorizzazione e l'ottenimento di serie temporali.

Tabella 3: Tecnologie per la gestione di dati temporali

2.4 - Tecnologie per la comunicazione e messaggistica

Tecnologia	Versione	Descrizione
NATS	2.10.25	NATS è un sistema di messaggistica open-source progettato per la comunicazione scalabile, affidabile e a bassa latenza tra servizi distribuiti. Supporta il pub/sub, request/reply e message queueing, rendendolo adatto a microservizi. Grazie alla sua leggerezza e semplicità, NATS permette un'elevata efficienza nella gestione della comunicazione tra componenti, garantendo resilienza e facilità di scalabilità senza necessità di configurazioni complesse.

Tabella 4: Tecnologie per la comunicazione e messaggistica

2.5 - Tecnologie per la virtualizzazione e *deployment*

Tecnologia	Versione	Descrizione
Docker Engine	28.0.1	Docker è una piattaforma di virtualizzazione che consente di impacchettare applicazioni e le loro dipendenze in container leggeri e portabili. Grazie alla sua architettura basata su immagini e container, Docker permette di garantire consistenza tra ambienti di sviluppo, test e produzione, semplificando il deployment e la scalabilità delle applicazioni. È particolarmente utile per microservizi e sistemi distribuiti, migliorando l'efficienza nell'uso delle risorse e la velocità di distribuzione del software.

Tabella 5: Tecnologie per la virtualizzazione e *deployment*

2.6 - Librerie

Tecnologia	Versione	Descrizione
zap	1.27.0	Libreria realizzata da Uber per il logging. Per maggiori informazioni si rimanda alla pagina Go del progetto
zap-prettyconsole	0.5.2	Libreria che rende maggiormente leggibile l'output di zap. Per maggiori informazioni si rimanda alla pagina Go del progetto
golang-jwt	5.2.2	Libreria utilizzata per generare e verificare i Token JWT. Per maggiori informazioni si rimanda alla pagina Go del progetto
google uuid	1.6.0	Libreria utilizzata per generare UUID, utili per identificare in maniera unica i vari servizi offerti dal Sistema. Per maggiori informazioni si rimanda alla pagina Go del progetto
jwx	1.2.30	Libreria utilizzata per il parsing delle chiavi pubbliche nel rese disponibili nell'apposito *subject* JetStream NATS. Per

		maggiori informazioni si rimanda alla pagina Go del progetto
nats	1.40.1	Client Go per l'utilizzo di NATS. Per maggiori informazioni si rimanda alla pagina Go del progetto
OpenTelemetry-Go	1.35.0	Implementazione Go per OpenTelemetry: viene utilizzata per raccogliere dati dai vari microservizi del Sistema. Per maggiori informazioni si rimanda alla pagina Go del progetto
OpenTelemetry-otelzap	0.10.0	Libreria che permette di legare zap a OpenTelemetry-Go. Per maggiori informazioni si rimanda alla pagina Go del progetto
influxdb-client-go	2.14.0	Libreria per dialogare con il *Database* InfluxDB, un tipo di Base di Dati specificatamente realizzata per gestire serie temporali, come quelle che riguardano le notifiche. Per maggiori informazioni si rimanda alla pagina Go del progetto

Tabella 6: Librerie utilizzate

2.7 - Tecnologie per il monitoraggio dei microservizi

Tecnologia	Versione	Descrizione
Grafana	11.6.0	Grafana è una piattaforma open-source per la visualizzazione e l'analisi di dati di monitoraggio. Supporta diverse fonti di dati (come Prometheus, Loki e Mimir) e consente la creazione di dashboard interattive per il monitoraggio in tempo reale.
Prometheus	3.2.1	Prometheus è un sistema di monitoraggio e allerta open-source focalizzato sulla raccolta di metriche attraverso un modello pull.
Loki	3.4.2	Loki è un sistema di log aggregation sviluppato da Grafana Labs, ottimizzato per la gestione dei log in modo scalabile ed efficiente. Si integra con Grafana per la visualizzazione e utilizza un'architettura simile a Prometheus, semplificando la correlazione tra metriche e log.

Tabella 7: Tecnologie per il monitoraggio dei microservizi

2.8 - Tecnologie per analisi statica

Tecnologia	Versione	Descrizione
gocyclo	0.6.0	Strumento utile per il calcolo della complessità ciclomatica. Per maggiori informazioni si rimanda alla pagina Go del progetto

staticcheck	0.6.0	Per maggiori informazioni si rimanda alla pagina Go del progetto
lint	2.0.2	Si tratta di una utility utile nell'ambito della *Continuous Integration* per eseguire i vari *linter* da esso stesso inclusi ed alcuni esterni come staticcheck. Per maggiori informazioni si rimanda alla pagina GitHub del progetto

Tabella 8: Tecnologie per analisi statica

2.9 - Tecnologie per l'analisi dinamica

Tecnologia	Versione	Descrizione
go test	1.24.0	Si tratta di una componente parte della libreria standard di Go per la realizzazione di test. Per maggiori informazioni si rimanda alla pagina Go del progetto
GoMock	0.5.0	GoMock è un *framework* utilizzato per realizzare Mock che implementano interfacce: questo risulta particolarmente utile nella realizzazione di test di unità. Deprecato da Google, adesso è sviluppato da Uber: per maggiori informazioni si rimanda alla pagina Go del progetto
magiconair properties assert	1.8.9	La libreria offre funzionalità per scrivere file con proprietà, ma offre anche strumenti utili per realizzare testing ed è per questo scopo che viene utilizzata. Per maggiori informazioni si rimanda alla pagina Go del progetto
stretchr testify	1.10.0	Libreria specificatamente realizzata per permettere una più semplice realizzazione dei test, offrendo funzionalità per richiedere che un risultato sia di un certo tipo (esempio `nil`). Per maggiori informazioni si rimanda alla pagina Go del progetto
nats-server	2.11.0	Libreria utilizzata per permettere la realizzazione di un Server NATS Mock utilizzabile nei test di unità e di integrazione. Per maggiori informazioni si rimanda alla pagina Go del progetto

Tabella 9: Tecnologie per l'analisi dinamica

3 - Architettura

3.1 - Architettura logica

Il sistema è progettato seguendo l'**architettura esagonale**, un modello che promuove una netta separazione tra la logica di *business* e le interazioni con servizi esterni, fonti di dati e interfacce utente.
Questo approccio organizza il sistema attorno a un nucleo centrale, circondato da porte che fungono da interfacce con il mondo esterno, garantendo modularità e testabilità.

Il **nucleo** dell'applicazione contiene la logica di dominio e le regole di *business*, progettato per essere indipendente dai dettagli tecnologici esterni, in modo da favorire la manutenibilità e l'estendibilità del sistema.

Le **porte** costituiscono il punto di connessione tra il nucleo e il mondo esterno, consentendo una comunicazione strutturata:

- *Inbound Ports* (o *Use Cases*): consentono l'invocazione della logica del nucleo da parte di componenti esterni, definendo i punti di accesso all'applicazione e isolando la logica di dominio da implementazioni tecnologiche specifiche.
- *Outbound Ports*: permettono al nucleo di interagire con servizi esterni, mantenendo un'astrazione che preserva l'indipendenza della logica di business dai dettagli di implementazione.

I ***services*** implementano le inbound ports e fanno parte della business logic, concentrandosi esclusivamente sulla logica di dominio senza dipendenze tecnologiche specifiche.

Gli ***adapters*** rappresentano lo strato esterno del sistema e si suddividono in:

- *Input Adapters* (o *Controllers*): ricevono input dall'esterno e invocano le operazioni sulle porte in ingresso, traducendo le richieste esterne in operazioni comprensibili per il nucleo.
- *Output Adapters*: gestiscono la comunicazione con l'esterno attraverso le porte in uscita, traducendo le risposte del nucleo in formati comprensibili per i servizi esterni.

3.2 - Architettura di *deployment*

3.2.1 - Sistema a microservizi

L'architettura di deployment[G] adottata per il sistema è basata su **microservizi**, come richiesto dal capitolato[G].
Questa scelta consente una maggiore scalabilità, resilienza e indipendenza nello sviluppo e nel *deployment* dei componenti software.

Ogni microservizio è indipendente e responsabile[G] di un insieme specifico di funzionalità[G].

I microservizi comunicano tra loro tramite NATS[G], un sistema di messaggistica publish-subscribe ad alte prestazioni. Questa soluzione permette:

- Comunicazione asincrona, sincrona ed *event-driven*, riducendo l'accoppiamento tra i servizi;
- Maggiore scalabilità, in quanto i messaggi possono essere gestiti in parallelo;
- Affidabilità nella trasmissione dei dati grazie alla capacità di gestire il *buffering* e il re-invio dei messaggi in caso di errore.

Oltre a NATS[G], i microservizi possono esporre API[G] REST per le comunicazioni con il *client*.

Il *deployment* dei microservizi avviene in ambienti virtualizzati tramite Docker[G]. Questo garantisce:

- Scalabilità dinamica, adattando le risorse ai carichi di lavoro;
- Isolamento dei servizi, evitando impatti negativi tra componenti;
- Gestione semplificata del ciclo di vita dei servizi.

Questa architettura[G] consente di ottenere un sistema altamente scalabile, resiliente e facilmente manutenibile, ottimizzato per ambienti distribuiti e carichi di lavoro variabili.

3.3 - Design pattern

3.3.1 - Dependency injection

3.3.1.1 - Descrizione del pattern

Il pattern *Dependency Injection*[G] è un design pattern strutturale che consente di rendere esplicite le dipendenze di un oggetto.
Invece di creare direttamente le dipendenze all'interno delle classi o dei componenti, queste possono essere fornite dall'esterno: in questo modo, un componente dichiara le sue dipendenze senza doversi preoccupare di istanziarle, permettendo dunque una maggiore modularità tra i diversi componenti del Sistema. Esistono principalmente due tipi di *dependency injection*[G]:

- *Constructor Injection*: le dipendenze vengono passate attraverso il costruttore;
- *Setter Injection*: le dipendenze vengono impostate tramite metodi setter.

Nel progetto viene utilizzata la **Constructor Injection**.

3.3.1.2 - Motivazioni dell'utilizzo del pattern

L'utilizzo del pattern *Dependency Injection*[G] nel progetto porta numerosi vantaggi:

- **disaccoppiamento**: i componenti sono meno legati tra loro, facilitando la manutenzione[G] e l'estensione del codice;
- **flessibilità**: è più semplice sostituire un'implementazione con un'altra senza modificare il codice client;
- **testabilità**: è possibile sostituire le istanze reali degli oggetti richiesti con *mock* durante i test[G]
- **modularità**: i componenti possono essere sviluppati, testati e utilizzati in modo indipendente;
- **gestione centralizzata**:la *dependency injection*[G] rende possibile l'utilizzo di *framework* specifici per la fornitura automatica delle istanze necessarie per soddisfare le dipendenze (nel nostro caso, attraverso **Fx**, vedi Sezione 3.3.1.3).

3.3.1.3 - Framework Fx di *Uber*

Fx è un *framework* per la *dependency injection*[G] utilizzabile con il linguaggio **Go**[G] sviluppato da *Uber*.
Per maggiori informazioni si consiglia la consultazione del sito web del progetto.

3.3.1.3.1 - Costrutti principali

I costrutti principali di Fx utilizzati nel progetto sono:

- **fx.Provide**: registra una funzione costruttore, ovvero una funzione che crea e restituisce un'istanza di un oggetto. Tale istanza verrà poi riutilizzata ovunque sarà richiesta come dipendenza;
- **fx.Supply**: fornisce valori istanziati per *Dependency Injection*[G] come se fossero stati restituiti da un costruttore.

Viene utilizzato il tipo più specifico di ogni valore;

- **fx.Invoke**: esegue una funzione dopo la costruzione di tutte le dipendenze;
- **fx.Options**: raggruppa più opzioni Fx, quindi dipendenze già dichiarate, garantendo una configurazione modulare, rapida e flessibile.
- **fx.Lifecycle**: gestisce il ciclo di vita dell'applicazione con hook OnStart e OnStop;
- **fx.In**: sono espressioni da inserire all'interno di strutture con un insieme di dipendenze da fornire ad un oggetto. Molto utile per ridurre il numero di parametri richiesti da un costruttore.

Un aspetto importante del progetto è l'uso di file *.module.go in cui vengono definiti moduli **Fx** per organizzare le dipendenze in modo gerarchico.
Ogni modulo espone una variabile *Module* che aggrega tutte le opzioni Fx relative a quel componente mediante fx.Options.

3.3.1.4 - Utilizzo del pattern nel progetto

Nel progetto, il pattern *Dependency Injection*[G] viene applicato in modo estensivo, utilizzandolo praticamente in ogni componente dell'architettura. Ogni microservizio utilizza **Fx** per gestire le proprie dipendenze, dalla configurazione fino ai componenti applicativi. I servizi vengono costruiti dinamicamente all'avvio dell'applicazione, con tutte le dipendenze iniettate automaticamente mediante **Fx**.

3.3.2 - Object adapter

3.3.2.1 - Descrizione del pattern

Il pattern *Object Adapter* è un design pattern di tipo strutturale che permette ad oggetti con interfacce incompatibili di collaborare tra loro.
In particolare, il *client* comunica con un'interfaccia *target* implementata dall'*adapter*, il quale, tramite composizione, contiene l'oggetto *adaptee*.
L'adapter agisce come un intermediario, convertendo le richieste del client in chiamate compatibili con l'oggetto adattato. A differenza del *Class Adapter* (che utilizza l'ereditarietà), l'*Object Adapter* mantiene un riferimento all'oggetto da adattare (*adaptee*), delegando a quest'ultimo l'esecuzione della richiesta appena convertita. La struttura tipica dell'Object Adapter include:

- una *Target Interface* che definisce l'interfaccia che il client si aspetta di utilizzare;
- un *Adaptee* che è l'oggetto con l'interfaccia incompatibile;
- un *Adapter* che implementa l'interfaccia Target e mantiene un riferimento all'*Adaptee*.

3.3.2.2 - Motivazioni dell'utilizzo del pattern

L'utilizzo del pattern *Object adapter* nel progetto porta diversi vantaggi:

- **flessibilità maggiore**: l'*adapter* può lavorare con qualsiasi sottoclasse dell'oggetto adattato;

- **riutilizzabilità**: l'*adapter* può adattare diverse interfacce;
- **manutenibilità**: separando l'*adapter* dall'oggetto concreto, eventuali modifiche all'oggetto non impattano direttamente sull'*adapter*.

3.3.2.3 - Utilizzo del pattern nel progetto

Nel contesto dell'architettura esagonale adottata, gli *Object Adapter* sono utilizzati principalmente tra gli oggetti che assolvono i compiti della *business logic* e della *persistence logic*, ma anche per le comunicazioni dalla *business logic* all'esterno del microservizio.

- *Input Adapter* (o *Controller*): implementati come *adapter* che traducono le richieste esterne (come richieste HTTP o messaggi NATS[G])in chiamate alle *inbound ports* (o *Use Cases*) del nucleo applicativo. Questi *adapter* convertono i formati di richiesta esterni in oggetti di dominio comprensibili per il nucleo;
- *Output Adapter*: implementano le *outbound ports* e adattano le chiamate del nucleo verso servizi esterni come sistemi di messaggistica o API[G] di terze parti. Questi *adapter* convertono gli oggetti di dominio in formati compatibili con i sistemi esterni. Un esempio è **AuthAdapter**, le cui informazioni sono disponibili alla Sezione 3.4.7.8.

3.3.3 - Strategy

3.3.3.1 - Descrizione del pattern

Il pattern *Strategy* consiste nell'incapsulare una parte di codice all'interno di una classe specifica. Tale classe deve soddisfare un'interfaccia che viene poi utilizzata dal Sistema per invocarne i metodi esposti: questo pattern permette di cambiare in maniera semplice e veloce parti di codice soggette a modifiche.

3.3.3.2 - Motivazioni dell'utilizzo del pattern

Il pattern *Strategy* viene utilizzato quando una componente di natura algoritmica del Sistema sviluppato non è definitiva ma soggetta a cambiamenti futuri: eseguirne la codifica forzata all'interno della funzione che la utilizza potrebbe risultare controproducente, in quanto, un suo cambiamento, potrebbe determinare uno stravolgimento del codice: per questo motivo, la parte algoritmica viene incapsulata all'interno di un oggetto che può essere facilmente sostituito.

3.3.3.3 - Utilizzo del pattern nel progetto

Nel nostro progetto, il pattern *Strategy* è stato utilizzato nel microservizio *Authenticator* (Sezione 3.4.7): M31 infatti necessita sicuramente in futuro di rendere più complesso l'algoritmo utilizzato per verificare le credenziali di un Utente, aggiungendo anzitutto il controllo di *username* e *password* e altri meccanismi in un secondo momento. Ogni algoritmo, per via della maggiore complessità, potrebbe avere sensibili differenze e nessun tratto in comune con le versioni precedenti, motivo per cui si è deciso di utilizzare questo pattern e non il pattern *Template Method*.

Nel progetto, il pattern *Strategy* è stato applicato per il calcolo della disponibilità delle merci nei magazzini. L'interfaccia `ICalculateAvailabilityUseCase` definisce il contratto per tutte le implementazioni che calcolano la disponibilità. Attualmente, è presente un'unica implementazione di questa interfaccia, denominata `SimpleCalculateAvailabilityService`. Questa implementazione utilizza un algoritmo base per determinare la disponibilità delle merci, considerando i dati forniti dai magazzini. Future implementazioni potrebbero

includere algoritmi più complessi, come quelli basati su intelligenza artificiale o machine learning, per ottimizzare ulteriormente il calcolo della disponibilità.

3.4 - Microservizi sviluppati

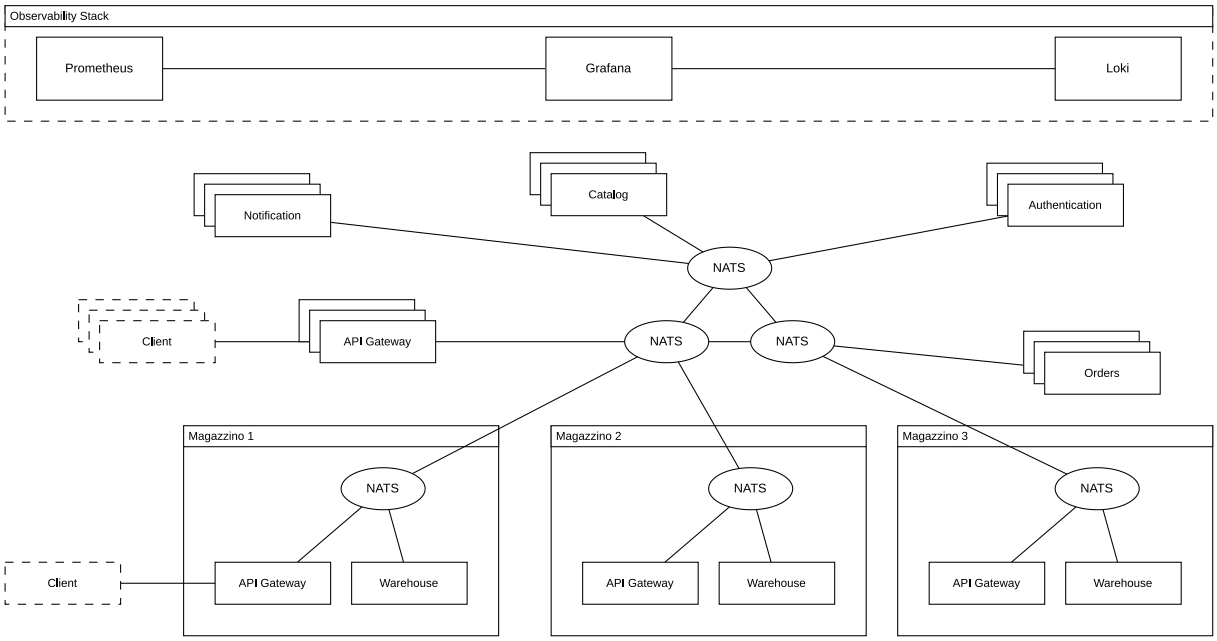

Figura 1: Microservizi inclusi nel Sistema

Sebbene **Go**[G] non abbia il concetto di «classe», comunque è possibile realizzare **strutture** e **funzioni** invocabili solo da quelle specifiche strutture, potendo così imitare, nel senso largo del termine, il funzionamento delle classi in un linguaggio di programmazione ad oggetti.

Si noti come dunque, in questo documento, i termini struttura e classe saranno utilizzati come sinonimi per il motivo sopra citato.

Per via del linguaggio utilizzato, talvolta potrebbe non essere stato possibile utilizzare il concetto di *Information Hiding*, specie in caso l'oggetto in questione abbia necessità di essere serializzato in formato *Json*. Inoltre, nel linguaggio Go[G], la privatezza di un attributo resta comunque limitata al di fuori del *package*.

Se la descrizione di un oggetto è assente questo implica che tale oggetto è una **struttura vuota**, ovvero senza alcun attributo e funzione da lei solamente invocabile.

Infine, si ricorda che, nel linguaggio Go[G], un nome di funzione o attributo che inizia con lettera minuscola simboleggia che lo stesso è visibile solo all'interno dello stesso *package*.

3.4.1 - Telemetria

I file dei vari `controller` e `listener` sono dotati di logger zap e di «contatori», oggetti di tipo `metric.Int64Counter`, come variabili globali. Nel costruttore dei `controller` e `listener` quest'ultime variabili vengono inizializzate con la funzione `CounterSetup` fornita dal package `observability`: tale funzione, la cui firma è `CounterSetup(meter *metric.Meter, logger *zap.Logger, counter *metric.Int64Counter, counterMap *sync.Map, name string, options ...metric.Int64CounterOption)` si occupa di prendere il logger del microservizio e la variabile da inizializzare, quindi l'oggetto `metric.Meter`, fornito nel costruttore del `controller` o `listener` e passato a questa funzione, viene sfruttato per creare un oggetto `metric.Int64Counter` che viene assegnato alla variabile `counter` passata come puntatore. Per evitare duplici assegnazioni, che potrebbero modificare il valore reale del contatore, una mappa `sync.Map` viene utilizzata per determinare se un contatore dello stesso tipo era già stato creato e assegnato.

La configurazione di **OpenTelemetry**, sfruttato da **Prometheus** e **Loki**, viene poi realizzata dal metodo `New` dello stesso *package*, che invoca la funzione `setupOtel` atta allo scopo di configurare il servizio.

Successivamente, per incrementare il numero di richieste, queste vengono incrementate dagli *Handler* dei *controller* e *listener* prima di terminare l'esecuzione, sfruttando il metodo `Add` degli oggetti `metric.Int64Counter`.

3.4.2 - Router dei microservizi

Ogni microservizio fa uso di NATS[G] ,un *message broker* per la trasmissione dei messaggi. Contrariamente ai linguaggi di programmazione come **Java** o **Kotlin**, **Go**[G] non prevede le annotazioni, strumenti utili in questi contesti poiché permettono la generazione automatica di codice che potrebbe, nel caso in questione, gestire la ricezione di un messaggio dalla rete NATS[G] e automaticamente convertirlo in un'istanza di un DTO richiesto dal metodo del controller adibito alla gestione di una determinata richiesta.

I vari microservizi utilizzano dunque codice esplicito che esegue questo processo, dalla registrazione del microservizio all'apposito canale di comunicazione NATS[G] ,sino a richiamare il giusto metodo del controller che deve gestire una specifica richiesta: la conversione al DTO trasmesso viene effettuata internamente alla funzione richiamata.

Nello specifico, ogni microservizio possiede:

- dei **Router**: tale struttura ha il compito, mediante la funzione `Setup`, di registrare il controller con un *subject* NATS[G] o JetStream NATS[G] ,associandovi la funzione da eseguire all'arrivo di un nuovo messaggio
- dei **Controller/Listener Router**: si tratta di una componente che possiede più `Router` e un metodo `Setup`. Dal momento che un microservizio potrebbe avere più controller e dunque più router, questa componente prende tutti i router del microservizio e vi invoca `Setup`. Per maggiori informazioni sul suo impiego, si veda la Sezione 3.4.4.

3.4.2.1 - Esempio: catalogRouter

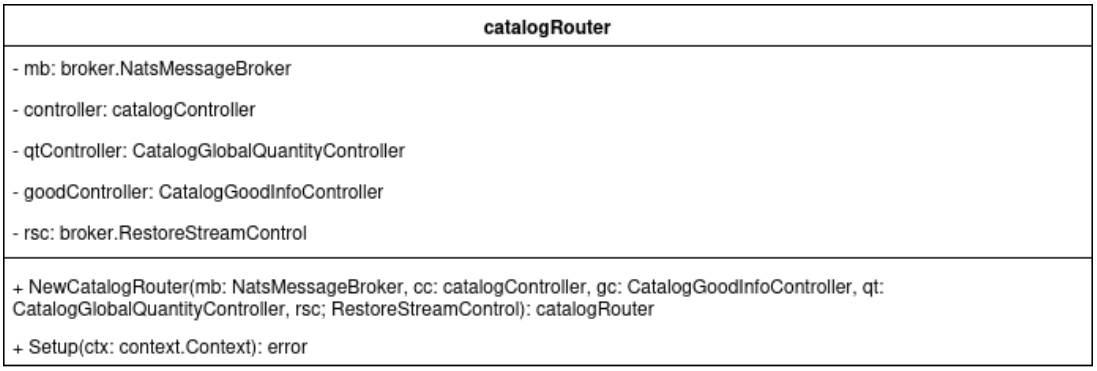

Figura 2: CatalogRouter

catalogRouter, router del microservizio **Catalog**, possiede i seguenti attributi:

- **`NatsMessageBroker`**: si occupa della completa gestione della connessione a NATS[G] e JetStream NATS[G] ,nonché possiede un logger per l'integrazione con **Grafana**[G] .Esso infatti possiede i seguenti attributi:
  - **`Nats *nats.Conn`**: connessione a NATS[G] ;
  - **`Js jetstream.JetStream`**: struttura per gestire connessioni a JetStream NATS[G] ;
  - **`Logger *zap.logger`**: logger.

E può invocare le seguenti funzioni:
  - **`NewNatsMessageBroker(nc *nats.Conn, logger *zap.Logger) (*NatsMessageBroker, error)`**: costruttore della struttura;
  - **`RegisterRequest(ctx context.Context, subject Subject, queue Queue, handler RequestHandler) error`**: permette di associare una funzione di un controller come *handler* di un messaggio in arrivo sul *subject* specificato;
  - **`RegisterJsHandler(ctx context.Context, restore IRestoreStreamControl, streamCfg jetstream.StreamConfig, handler JsHandler, opts ...JsHandlerOpt) error`**: permette di associare una funzione di un controller come *handler* di un messaggio in arrivo sul *subject* (del JetStream NATS[G])specificato;
  - **`RegisterJsWithConsumerGroup(ctx context.Context, streamCfg jetstream.StreamConfig, consumerCfg jetstream.ConsumerConfig, handler JsHandler) error`**: come il precedente, ma permette di applicare un gruppo di *consumer* (non viene utilizzato da questo router).
- **`controller *catalogController`**: il controller del microservizio **Catalog** dedicato alla gestione dei magazzini e della quantità della merce;
- **`goodController *CatalogGoodInfoController`**: il controller del microservizio **Catalog** dedicato alla gestione delle informazioni della merce;
- **`qtController *CatalogGlobalQuantityController`**: il controller del microservizio **Catalog** dedicato alla gestione delle richieste di ottenimento quantità globale della merce;
- **`rsc *broker.RestoreStreamControl`**: una struttura che fa uso di `sync.WaitGroup` per gestire il recupero dei messaggi dai JetStream. È infatti necessario per l'invocazione di metodi quali `RegisterJsHandler`.

Può invocare le seguenti funzioni:

- **`NewCatalogRouter(mb *broker.NatsMessageBroker, cc *catalogController, gc *CatalogGoodInfoController, qt *CatalogGlobalQuantityController, rsc *broker.RestoreStreamControl) *catalogRouter`**: il costruttore del Router;
- **`Setup(ctx context.Context) error`**: esegue le associazioni *controller* - *subject* usando i metodi sopra descritti.

Per ulteriori informazioni in merito ai *subject* è possibile visionare quanto necessario direttamente nel codice, mentre le configurazioni dei vari JetStream NATS[G] sono disponibili nella cartella common/stream del *repository*[G] .

3.4.3 - Configurazioni dei microservizi

La configurazione dei vari microservizi, specie per l'indirizzo di accesso a NATS[G] e per la raccolta di dati telemetrici, può avvenire inserendo variabili di ambiente nel file `compose.yml`.

Per maggiori informazioni si rimanda alla lettura del Manuale Utente[G].

3.4.4 - `Main` dei microservizi

Ogni microservizio possiede il proprio `Main` che raccoglie i vari fil `*.mdoule.go` necessari all'esecuzione del microservizio e invoca la funzione di avvio denominata `RunLifeCycle`, il tutto utilizzando il *framework* fx.

Tale funzione prende solitamente il `ControllerRouter` e/o i `Listener` e li passa alla funzione `Run` che vi richiama il metodo Setup, perciò concretamente avviando il microservizio.

3.4.5 - Funzionamento Ordini e Trasferimenti

Ogni magazzino è gestito da un microservizio dedicato, responsabile della gestione dello stock[G] specifico di quel magazzino. La gestione degli ordini è invece affidata al microservizio Order (Sezione 3.4.8), che monitora costantemente gli aggiornamenti provenienti dai vari magazzini per mantenere aggiornato il proprio stato interno sulle disponibilità.

Quando il client ha terminato di costruire un ordine[G] localmente, per confermarlo contatta il microservizio Order, che genera un evento di tipo *order_update* con stato *Created*. Questo evento viene salvato nello stream di NATS[G]. Contemporaneamente, viene inviato un evento sullo stream *contact_warehouses*, che sarà ascoltato dai microservizi Order (Sezione 3.4.8) tramite un *consumer group*, in modo che una sola istanza gestisca l'evento. Questo microservizio si occupa di contattare i magazzini coinvolti per prenotare le merci necessarie. La prenotazione avviene tramite una richiesta NATS[G].

La selezione dei magazzini avviene in base alla disponibilità delle merci richieste, utilizzando un algoritmo che privilegia i magazzini con una quantità di merce più vicina a quella necessaria.

Dopo aver completato con successo la prenotazione delle merci presso i magazzini interessati, l'ordine viene aggiornato allo stato *Filled*. A questo punto, viene generato un nuovo evento di tipo *order_update*, che include la lista delle prenotazioni effettuate. I microservizi Warehouse[G] (Sezione 3.4.10) coinvolti ricevono questo evento e aggiornano di conseguenza lo stock[G] disponibile.

Infine, quando i microservizi Order ricevono gli aggiornamenti degli stock[G] dai magazzini, lo stato dell'ordine viene aggiornato internamente a *Completed*, informando così l'utente del completamento dell'ordine.

Per evitare conflitti tra ordini, il sistema utilizza, dunque, un meccanismo di prenotazioni che assicura la disponibilità delle merci al momento della conferma dell'ordine. Inoltre, la gestione distribuita degli stock[G] tra i vari magazzini elimina problemi di concorrenza e migliora la scalabilità complessiva del sistema.

Una funzionalità[G] simile agli ordini è quella dei trasferimenti, dei particolari ordini che permettono di spostare della merce da un magazzino mittente a un magazzino destinatario. I trasferimenti vengono gestiti in modo simile agli ordini, con la differenza che non viene utilizzato nessun algoritmo per la selezione del magazzino destinatario, in quanto è già specificato nella richiesta di trasferimento[G] ,e vengono utilizzati gli eventi di tipo *transfer_update* al posto degli omologhi per gli ordini (*order_update*).

3.4.6 - Oggetti comuni tra microservizi

3.4.6.1 - ResponseDTO

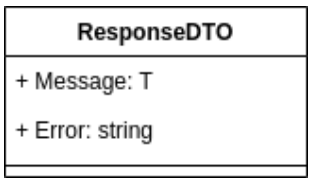

Figura 3: Common - ResponseDTO

Rappresenta un DTO generico per le risposte, utilizzabile per incapsulare un messaggio di tipo generico `T` e un eventuale messaggio di errore.

Descrizione degli attributi della struttura:

- **`Message T`**: rappresenta il messaggio generico di tipo `T` incluso nella risposta;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.2 - StockUpdate

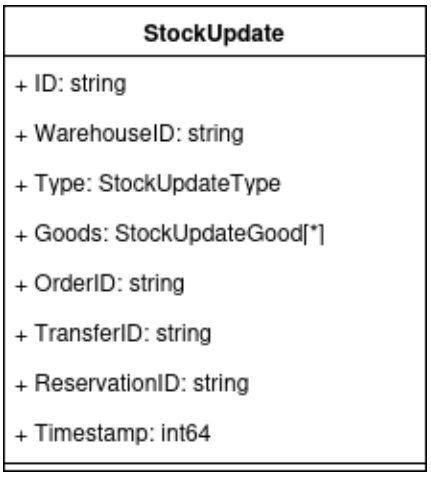

Figura 4: Warehouse - StockUpdate

Rappresenta un aggiornamento dello stock[G] nel sistema, utilizzato per comunicare variazioni di quantità di merci tra i microservizi.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'aggiornamento dello stock[G]
- **`WarehouseID string`**: rappresenta l'identificativo del magazzino coinvolto nell'aggiornamento dello stock[G]
- **`Type StockUpdateType`**: rappresenta il tipo di aggiornamento dello stock[G]. Può assumere i seguenti valori:
  - **`add`**: per aggiungere stock[G]
  - **`remove`**: per rimuovere stock[G]
  - **`order`**: per aggiornamenti legati a ordini;
  - **`transfer`**: per aggiornamenti legati a trasferimenti;
- **`Goods []StockUpdateGood`**: rappresenta una lista di oggetti `StockUpdateGood` che contengono le informazioni sulle merci aggiornate;
- **`OrderID string`**: rappresenta l'identificativo dell'ordine associato all'aggiornamento dello stock[G]
- **`TransferID string`**: rappresenta l'identificativo del trasferimento[G] associato all'aggiornamento dello stock[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata all'aggiornamento dello stock[G]
- **`Timestamp int64`**: rappresenta il *timestamp* dell'aggiornamento dello stock[G].

3.4.6.3 - StockUpdateType

Rappresenta il tipo di aggiornamento dello stock[G]. È un tipo stringa con i seguenti valori possibili:

- **`add`**: per aggiungere stock[G]
- **`remove`**: per rimuovere stock[G]
- **`order`**: per aggiornamenti legati a ordini;
- **`transfer`**: per aggiornamenti legati a trasferimenti.

3.4.6.4 - StockUpdateGood

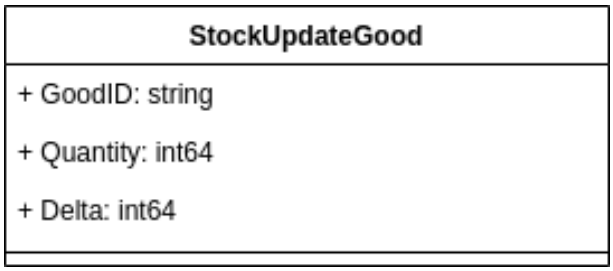

Figura 5: Warehouse - StockUpdateGood

Rappresenta una merce aggiornata nel comando di aggiornamento dello stock[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce aggiornata;
- **`Quantity int64`**: rappresenta la nuova quantità della merce aggiornata;
- **`Delta int64`**: rappresenta la differenza di quantità della merce rispetto all'ultimo stato.

3.4.6.5 - AddStockRequestDTO

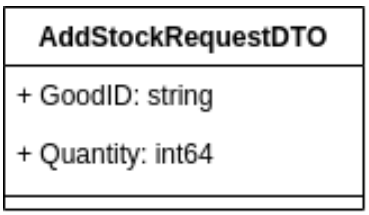

Figura 6: Warehouse - AddStockRequestDTO

Questo DTO viene utilizzato per rappresentare la richiesta di aggiunta di stock[G], e viene utilizzato dal controller Sezione 3.4.10.28

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'id della merce a cui aggiungere stock[G]
- **`Quantity int64`**: rappresenta la quantità di stock[G] da aggiungere alla merce.

3.4.6.6 - RemoveStockRequestDTO

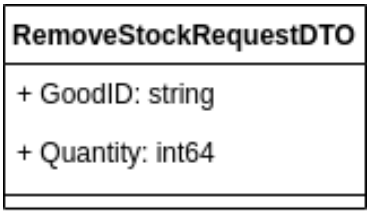

Figura 7: Warehouse - RemoveStockRequestDTO

Questo DTO viene utilizzato per rappresentare la richiesta di rimozione di stock[G], e viene utilizzato dal controller Sezione 3.4.10.28

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'id della merce da cui rimuovere stock[G]
- **`Quantity int64`**: rappresenta la quantità di stock[G] da rimuovere dalla merce.

3.4.6.7 - ReserveStockRequestDTO

Figura 8: Warehouse - ReserveStockRequestDTO

Questo DTO viene utilizzato per rappresentare la richiesta di prenotazione di stock[G] nel magazzino.

Descrizione degli attributi della struttura:

- **`Goods []ReserveStockItem`**: rappresenta una lista di oggetti `ReserveStockItem` che contengono le informazioni sulle merci da prenotare.

3.4.6.8 - ReserveStockItem

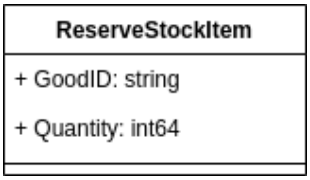

Figura 9: Warehouse - ReserveStockItem

Rappresenta una merce coinvolta nella richiesta di prenotazione di stock[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce da prenotare;
- **`Quantity int64`**: rappresenta la quantità della merce da prenotare.

3.4.6.9 - HealthCheckResponseDTO

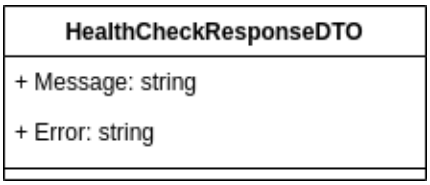

Figura 10: Warehouse - HealthCheckResponseDTO

Rappresenta un DTO specifico per le risposte di controllo dello stato di salute del microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message string`**: rappresenta il messaggio di stato del microservizio **Warehouse**[G]
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.10 - ReserveStockResponseDTO

Figura 11: Warehouse - ReserveStockResponseDTO

Rappresenta un DTO specifico per le risposte relative alla prenotazione di stock[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message ReserveStockInfo`**: rappresenta le informazioni sulla prenotazione dello stock[G] nel microservizio **Warehouse**[G]
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.11 - ReserveStockInfo

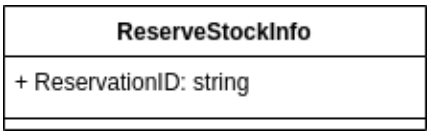

Figura 12: Warehouse - ReserveStockInfo

Rappresenta le informazioni relative a una prenotazione di stock[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`ReservationID string`**: rappresenta l'identificativo univoco della prenotazione nel microservizio **Warehouse**[G].

3.4.6.12 - OrderUpdate

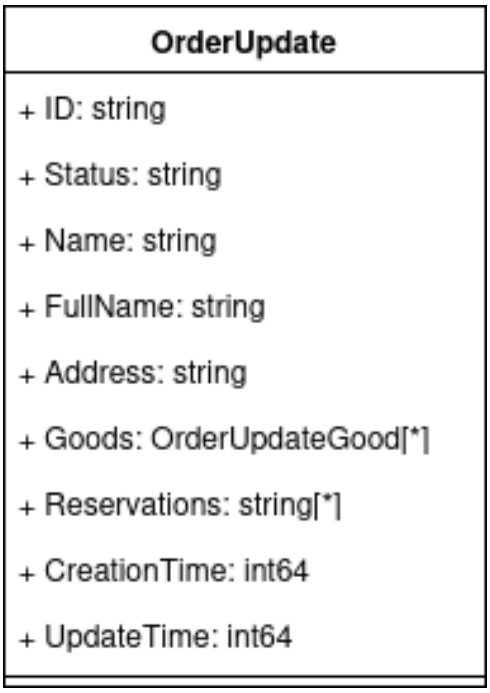

Figura 13: Order - OrderUpdate

Rappresenta un aggiornamento di un ordine[G] nel sistema.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine aggiornato;
- **`Status string`**: rappresenta lo stato dell'ordine aggiornato (es. «Created», «Filled»);
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []OrderUpdateGood`**: rappresenta una lista di oggetti `OrderUpdateGood` che contengono le informazioni sulle merci coinvolte nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine.

3.4.6.13 - OrderUpdateGood

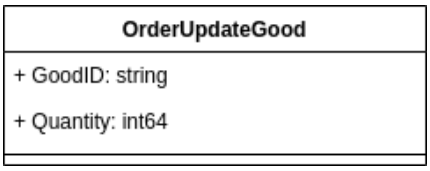

Figura 14: Order - OrderUpdateGood

Rappresenta una merce coinvolta in un aggiornamento di un ordine[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nell'ordine;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nell'ordine.

3.4.6.14 - TransferUpdate

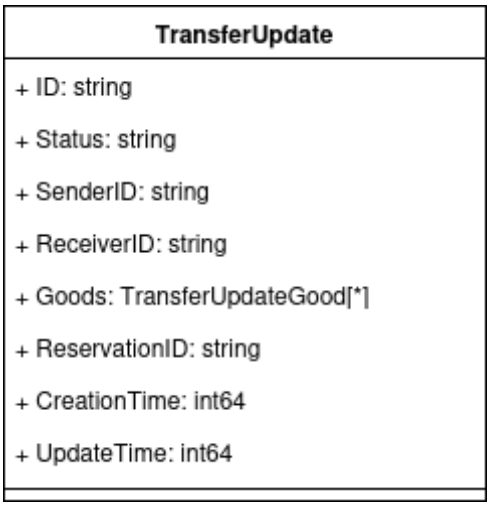

Figura 15: Order - TransferUpdate

Rappresenta un aggiornamento di un trasferimento[G] nel sistema.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del trasferimento[G] aggiornato;
- **`Status string`**: rappresenta lo stato del trasferimento[G] aggiornato (es. «Created», «Filled»);
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []TransferUpdateGood`**: rappresenta una lista di oggetti `TransferUpdateGood` che contengono le informazioni sulle merci coinvolte nel trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G]
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G].

3.4.6.15 - TransferUpdateGood

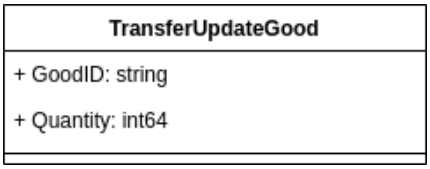

Figura 16: Order - TransferUpdateGood

Rappresenta una merce coinvolta in un aggiornamento di un trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nel trasferimento[G]
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nel trasferimento[G].

3.4.6.16 - CreateOrderRequestDTO

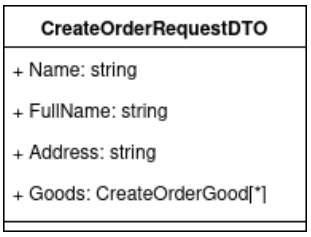

Figura 17: Order - CreateOrderRequestDTO

Rappresenta il DTO utilizzato per creare un nuovo ordine[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []CreateOrderGood`**: rappresenta una lista di oggetti `CreateOrderGood` che contengono le informazioni sulle merci incluse nell'ordine.

3.4.6.17 - CreateOrderGood

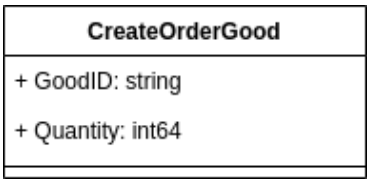

Figura 18: Order - CreateOrderGood

Rappresenta una merce inclusa in un ordine[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce inclusa nell'ordine;
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nell'ordine.

3.4.6.18 - CreateTransferRequestDTO

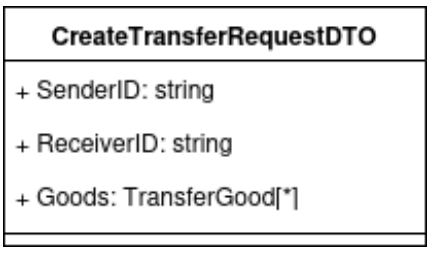

Figura 19: Order - CreateTransferRequestDTO

Rappresenta il DTO utilizzato per creare un nuovo trasferimento[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []TransferGood`**: rappresenta una lista di oggetti `TransferGood` che contengono le informazioni sulle merci incluse nel trasferimento[G].

3.4.6.19 - TransferGood

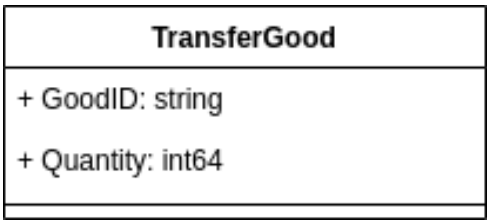

Figura 20: Order - TransferGood

Rappresenta una merce inclusa in un trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce inclusa nel trasferimento[G]
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nel trasferimento[G].

3.4.6.20 - GetOrderRequestDTO

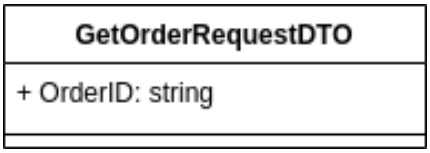

Figura 21: Order - GetOrderRequestDTO

Rappresenta il DTO utilizzato per richiedere i dettagli di un ordine[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`OrderID string`**: rappresenta l'identificativo univoco dell'ordine richiesto.

3.4.6.21 - GetTransferRequestDTO

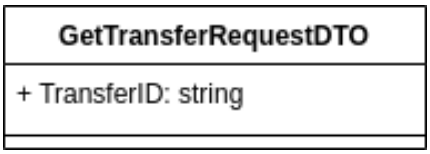

Figura 22: Order - GetTransferRequestDTO

Rappresenta il DTO utilizzato per richiedere i dettagli di un trasferimento[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`TransferID string`**: rappresenta l'identificativo univoco del trasferimento[G] richiesto.

3.4.6.22 - OrderCreateResponseDTO

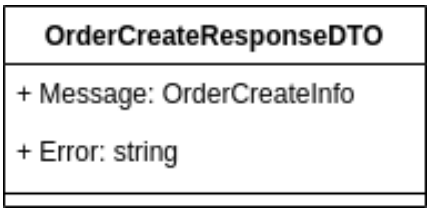

Figura 23: Order - OrderCreateResponseDTO

Rappresenta la risposta alla richiesta di creazione di un ordine[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message OrderCreateInfo`**: rappresenta le informazioni relative all'ordine creato;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.23 - OrderCreateInfo

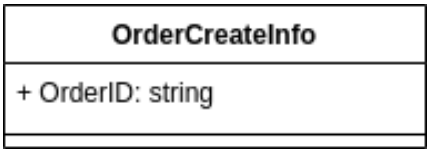

Figura 24: Order - OrderCreateInfo

Rappresenta le informazioni relative all'ordine creato nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`OrderID string`**: rappresenta l'identificativo univoco dell'ordine creato.

3.4.6.24 - GetOrderResponseDTO

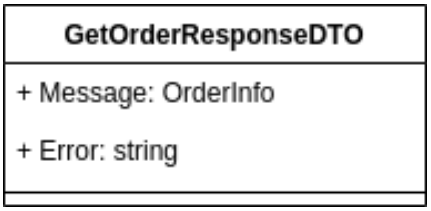

Figura 25: Order - GetOrderResponseDTO

Rappresenta la risposta alla richiesta di informazioni su un ordine[G] specifico nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message OrderInfo`**: rappresenta le informazioni relative all'ordine richiesto;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.25 - OrderInfo

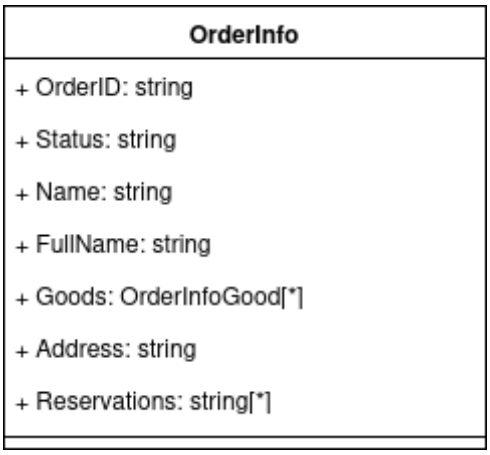

Figura 26: Order - OrderInfo

Rappresenta le informazioni relative a un ordine[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`OrderID string`**: rappresenta l'identificativo univoco dell'ordine;
- **`Status string`**: rappresenta lo stato dell'ordine;
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []OrderInfoGood`**: rappresenta una lista di oggetti `OrderInfoGood` che contengono le informazioni sulle merci incluse nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine.
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine.

3.4.6.26 - OrderInfoGood

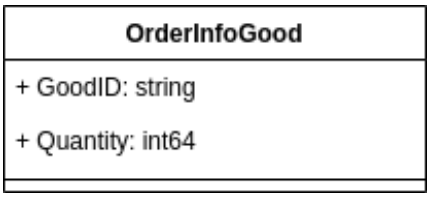

Figura 27: Order - OrderInfoGood

Rappresenta una merce inclusa in un ordine[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nell'ordine.

3.4.6.27 - GetAllOrderResponseDTO

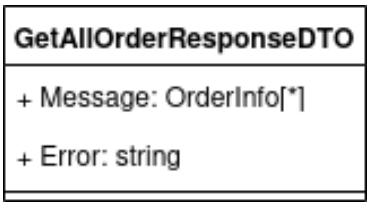

Figura 28: Order - GetAllOrderResponseDTO

Rappresenta la risposta alla richiesta di informazioni su tutti gli ordini nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message []OrderInfo`**: rappresenta una lista di informazioni relative agli ordini richiesti;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.28 - TransferCreateResponseDTO

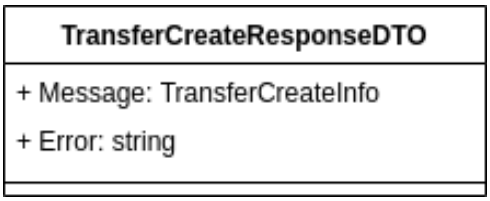

Figura 29: Order - TransferCreateResponseDTO

Rappresenta la risposta alla richiesta di creazione di un trasferimento[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message TransferCreateInfo`**: rappresenta le informazioni relative al trasferimento[G] creato;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.29 - TransferCreateInfo

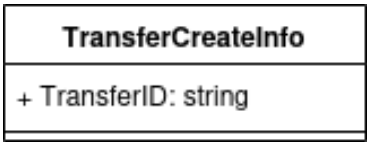

Figura 30: Order - TransferCreateInfo

Rappresenta le informazioni relative al trasferimento[G] creato nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`TransferID string`**: rappresenta l'identificativo univoco del trasferimento[G] creato.

3.4.6.30 - GetTransferResponseDTO

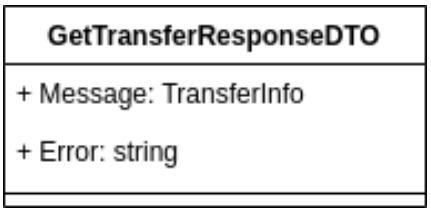

Figura 31: Order - GetTransferResponseDTO

Rappresenta la risposta alla richiesta di informazioni su un trasferimento[G] specifico nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message TransferInfo`**: rappresenta le informazioni relative al trasferimento[G] richiesto;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.31 - TransferInfo

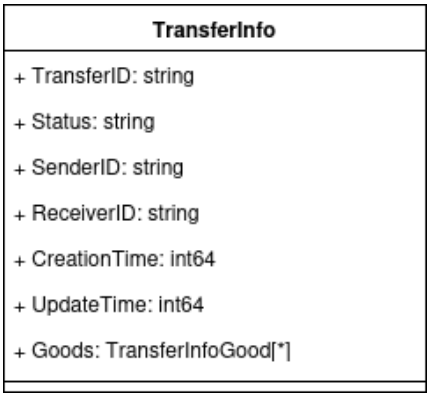

Figura 32: Order - TransferInfo

Rappresenta le informazioni relative a un trasferimento[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`TransferID string`**: rappresenta l'identificativo univoco del trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G]
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []TransferInfoGood`**: rappresenta una lista di oggetti `TransferInfoGood` che contengono le informazioni sulle merci incluse nel trasferimento[G].
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G].

3.4.6.32 - TransferInfoGood

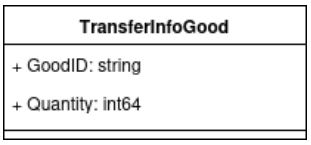

Figura 33: Order - TransferInfoGood

Rappresenta una merce inclusa in un trasferimento[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nel trasferimento[G].

3.4.6.33 - GetAllTransferResponseDTO

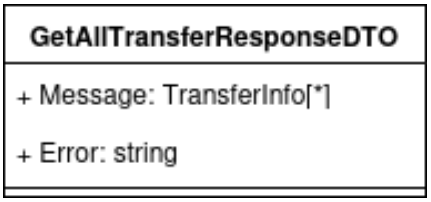

Figura 34: Order - GetAllTransferResponseDTO

Rappresenta la risposta alla richiesta di informazioni su tutti i trasferimenti nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`Message []TransferInfo`**: rappresenta una lista di informazioni relative ai trasferimenti richiesti;
- **`Error string`**: rappresenta un messaggio di errore, se presente.

3.4.6.34 - GoodUpdateData

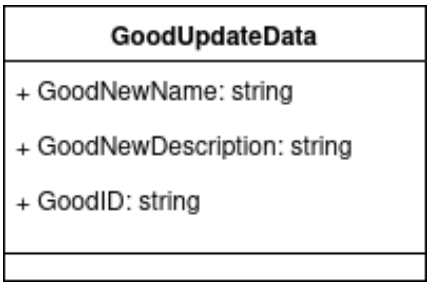

Figura 35: Catalog - GoodUpdateData

Struttura utile per rappresentare una modifica alle informazioni di una merce o all'aggiunta di una merce.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'id della merce da aggiungere o da modificare;
- **`GoodNewName string`**: rappresenta il nome da dare alla merce in questione;
- **`GoodNewDescription string`**: rappresenta la descrizione da dare alla merce in questione.

3.4.6.35 - GetGoodsDataResponseDTO

Figura 36: Catalog - GetGoodsDataResponseDTO

Rappresenta la risposta fornita alla richiesta di ottenimento informazioni sulle merci presenti nel Sistema.

Descrizione degli attributi della struttura:

- **`GoodMap map[string]catalogCommon.Good`**: è la mappa fornita in risposta alla richiesta. Come chiave ha una **string** che rappresenta l'id della merce, mentre come valore ha un oggetto descritto alla Sezione 3.4.9.1.2
- **`Err string`**: quando compilato, esplicita l'errore riscontrato nell'elaborare la richiesta. Se nessun errore è presente, la stringa è vuota.

3.4.6.36 - GetWarehouseResponseDTO

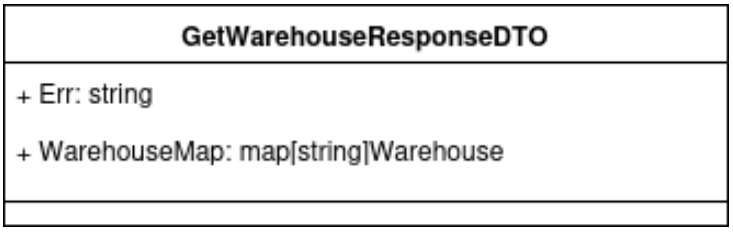

Figura 37: Catalog - GetWarehouseResponseDTO

Rappresenta la risposta alla richiesta di ottenimento informazioni sull'inventario dei magazzini memorizzati nel Sistema.

Descrizione degli attributi della struttura:

- **`WarehouseMap map[string]catalogCommon.Warehouse`**: mappa avente come chiave una **string** rappresentante l'id del magazzino, mentre valore un oggetto descritto alla Sezione 3.4.9.1.1
- **`Err string`**: quando compilato, esplicita l'errore riscontrato nell'elaborare la richiesta. Se nessun errore è presente, la stringa è vuota.

3.4.6.37 - GetGoodsQuantityResponseDTO

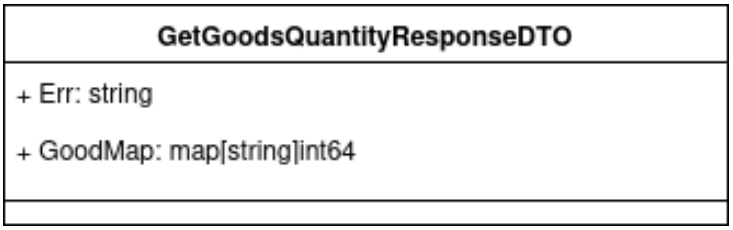

Figura 38: Catalog - GetGoodsQuantityResponseDTO

Rappresenta la risposta alla richiesta di ottenimento informazioni sulla quantità globale delle merci.

Descrizione degli attributi della struttura:

- **`GoodMap map[string]int64`**: mappa avente come chiave una **string** che rappresenta l'id della merce, mentre come valore un **int64** che ne rappresenta la quantità;
- **`Err string`**: quando compilato, esplicita l'errore riscontrato nell'elaborare la richiesta. Se nessun errore è presente, la stringa è vuota.

3.4.6.38 - AuthLoginRequest

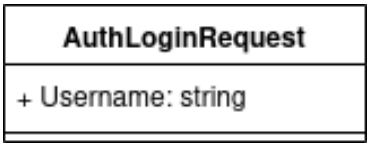

Figura 39: Authenticator - AuthLoginRequest

Rappresenta la richiesta di ottenimento di un Token, necessario per operare con il Sistema.

Descrizione degli attributi della struttura:

- **`Username string`**: username dell'utente che vuole ottenere un Token

3.4.6.39 - AuthLoginResponse

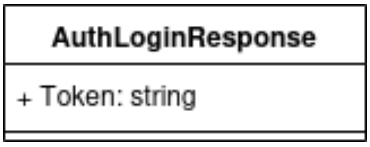

Figura 40: Authenticator - AuthLoginResponse

Rappresenta la risposta alla richiesta di un Token.

Descrizione degli attributi della struttura:

- **`token string`**: rappresenta il Token. Il campo rimane vuoto se la richiesta non era corretta.

3.4.7 - Authenticator

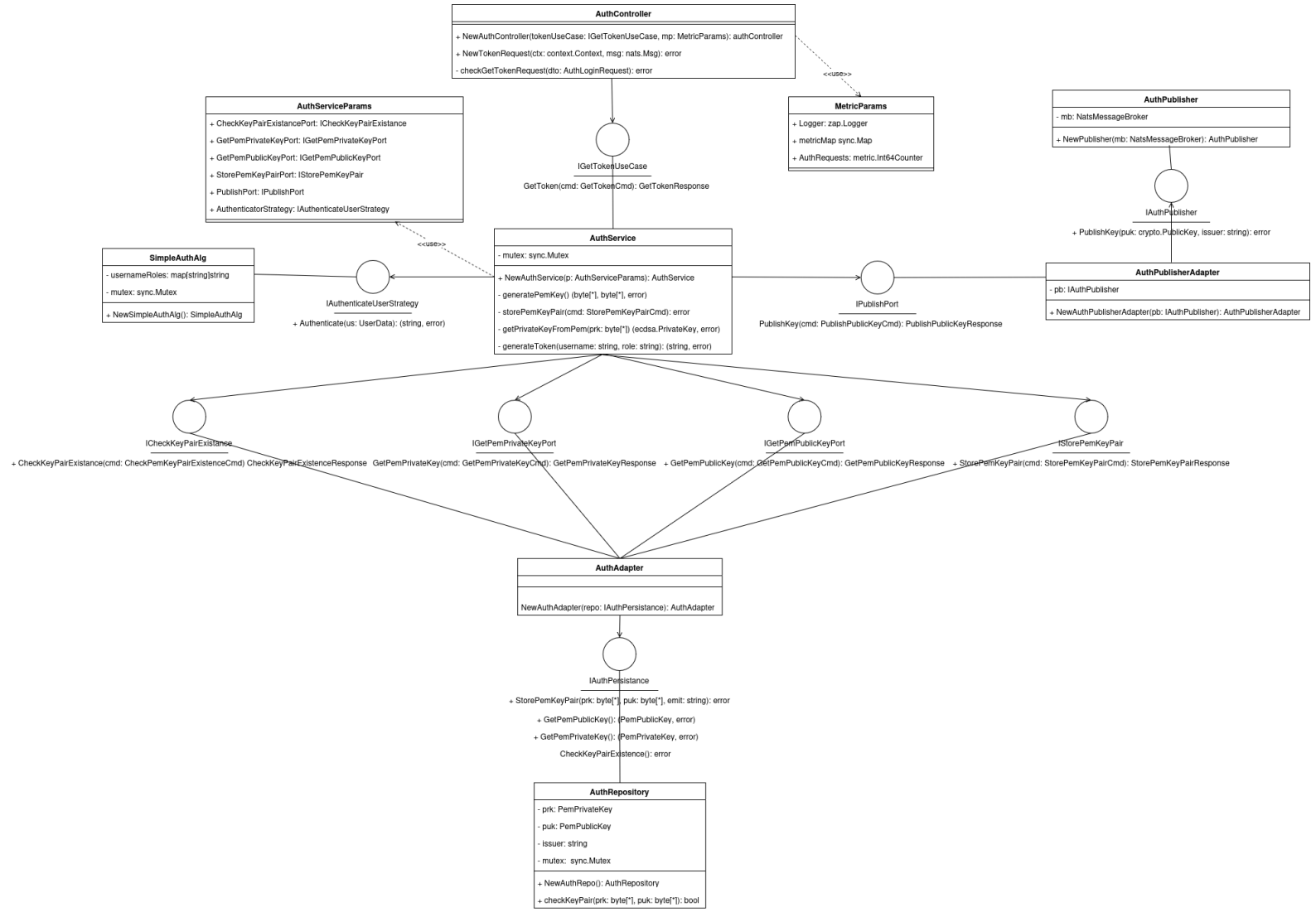

Figura 41: Authenticator - Authenticator

Il microservizio **Authenticator** si occupa di ricevere le richieste di ottenimento Token, controllarne i valori e restituire un Token valido e temporaneo (1 settimana di validità) affinché il *Client* possa utilizzare il Sistema.

I Token sono inoltre firmati con una chiave privata di tipo **ECDSA**, acronimo di *Elliptic Curve Digital Signature Algorithm*: la relativa chiave pubblica, necessaria per verificata i Token, viene pubblicata in un JetStream di NATS[G] ed è utilizzata dagli API[G] Gateway per verificare l'autenticità dei Token.

È formato dalle seguenti componenti:

- **AuthController**, che rappresenta l'*application logic*;
- **AuthService**, che rappresenta la *business logic*;
- **AuthRepository**, che rappresenta la *persistence logic*.

Le tre componenti, assieme agli oggetti eventualmente utilizzati saranno ora esposti.

3.4.7.1 - Oggetti comuni del microservizio

3.4.7.1.1 - CheckPemKeyPairExistenceCmd

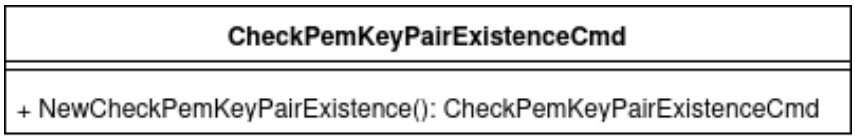

Figura 42: Authenticator - CheckPemKeyPairExistenceCmd

Rappresenta il *Command* per verificare l'esistenza della chiave pubblica e della chiave privata. Viene utilizzata dalla *business logic* per verificarne la presenza, necessaria per firmare il Token.

Descrizione degli attributi della struttura:

questa struttura non possiede attributi

Descrizione dei metodi invocabili dalla struttura:

- **`NewCheckPemKeyPairExistence() *CheckPemKeyPairExistenceCmd`**: rappresenta il costruttore del *Command*.

3.4.7.1.2 - GetPemPrivateKeyCmd

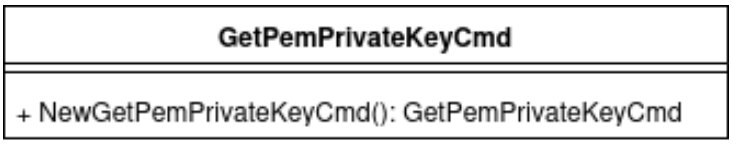

Figura 43: Authenticator - GetPemPrivateKeyCmd

Rappresenta il *Command* per ottenere la chiave privata, necessaria alla *business logic* per firmare il Token.

Descrizione degli attributi della struttura:

questa struttura non possiede attributi

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetPemPrivateKeyCmd() *GetPemPrivateKeyCmd`**: rappresenta il costruttore del *Command*.

3.4.7.1.3 - GetPemPublicKeyCmd

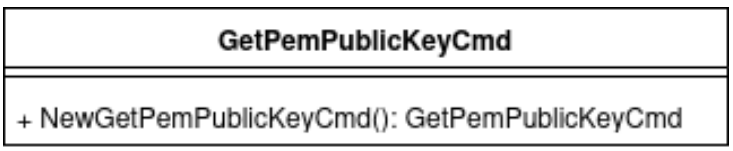

Figura 44: Authenticator - GetPemPublicKeyCmd

Rappresenta il *Command* per ottenere la chiave pubblica, necessaria alla *business logic* in quanto deve da questa essere pubblicata in un JetStream NATS[G] apposito.

Descrizione degli attributi della struttura:

questa struttura non possiede attributi

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetPemPublicKeyCmd() *GetPemPublicKeyCmd`**: rappresenta il costruttore del *Command*.

3.4.7.1.4 - GetTokenCmd

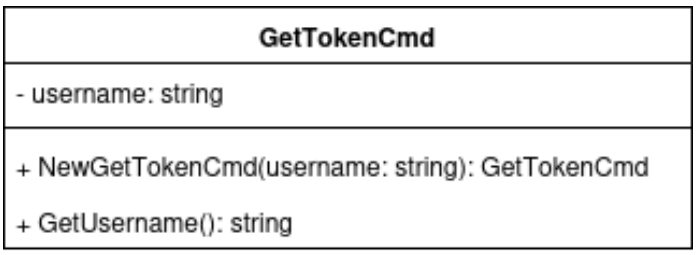

Figura 45: Authenticator - GetTokenCmd

Rappresenta il *Command* per ottenere la generazione di un Token.

Descrizione degli attributi della struttura:

- **`username string`**: rappresenta lo *username* dell'utente che ha richiesto il Token.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetTokenCmd(username string) *GetTokenCmd`**: rappresenta il costruttore del *Command*;
- **`GetUsername() string`**: permette di ottenere lo *username* registrato nel *Command*.

3.4.7.1.5 - PublishPublicKeyCmd

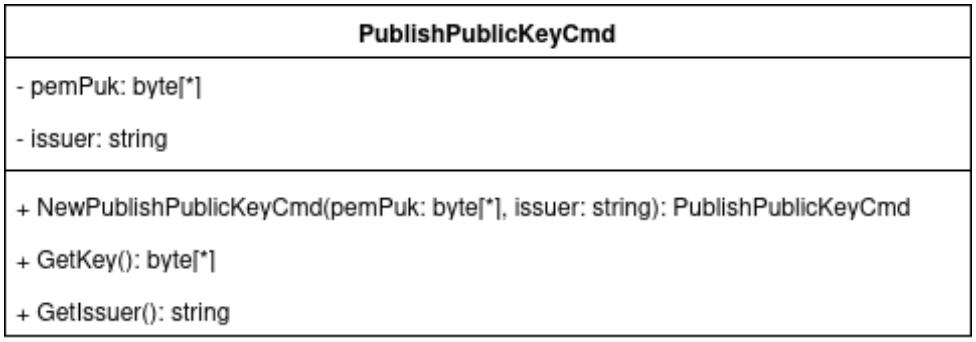

Figura 46: Authenticator - PublishPublicKeyCmd

Rappresenta il *Command* per ottenere la pubblicazione della chiave pubblica.

Descrizione degli attributi della struttura:

- **`pemPuk *[]byte`**: rappresenta la chiave pubblica, in formato Pem e memorizzata in byte;
- **`issuer string`**: rappresenta l'*issuer*, che la *business logic* genera al momento della generazione delle chiavi mediante la libreria uuid di Google.

Descrizione dei metodi invocabili dalla struttura:

- **`NewPublishPublicKeyCmd(pemPuk *[]byte, issuer string) *PublishPublicKeyCmd`**: rappresenta il costruttore del *Command*;
- **`GetKey() *[]byte`**: permette di ottenere la chiave registrata nel *Command*;
- **`GetIssuer() string`**: permette di ottenere lo *issuer* registrato nel *Command*.

3.4.7.1.6 - StorePemKeyPairCmd

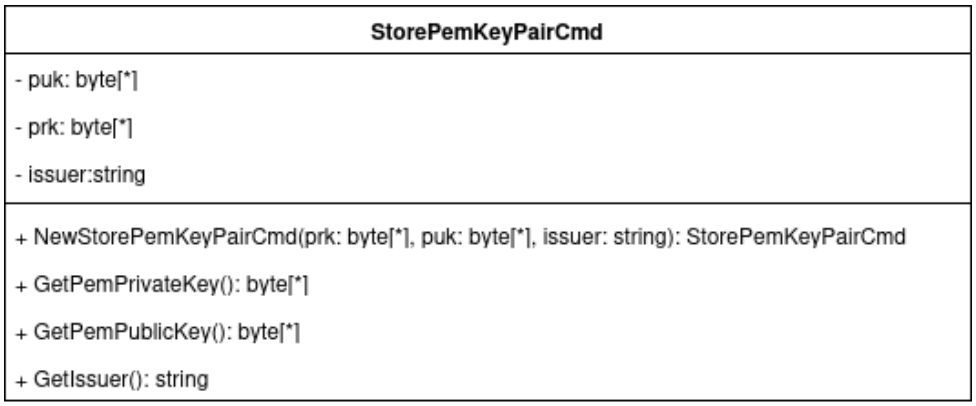

Figura 47: Authenticator - StorePemKeyPairCmd

Rappresenta il *Command* per ottenere la memorizzazione in *persistence logic* della coppia di chiavi generate dalla *business logic*.

Descrizione degli attributi della struttura:

- **`prk *[]byte`**: rappresenta la chiave privata, in formato Pem e memorizzata in byte;
- **`puk *[]byte`**: rappresenta la chiave pubblica, in formato Pem e memorizzata in byte;
- **`issuer string`**: rappresenta l'*issuer*, che la *business logic* genera al momento della generazione delle chiavi mediante la libreria uuid di Google.

Descrizione dei metodi invocabili dalla struttura:

- **`NewStorePemKeyPairCmd(prk *[]byte, puk *[]byte, issuer string) *StorePemKeyPairCmd`**: rappresenta il costruttore del *Command*;
- **`GetPemPrivateKey() *[]byte`**: permette di ottenere la chiave privata registrata nel *Command*;
- **`GetPemPublicKey() *[]byte`**: permette di ottenere la chiave privata registrata nel *Command*;
- **`GetIssuer() string`**: permette di ottenere lo *issuer* registrato nel *Command*.

3.4.7.1.7 - CheckKeyPairExistenceResponse

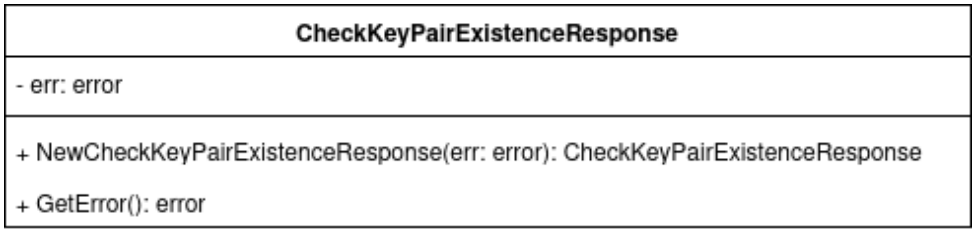

Figura 48: Authenticator - CheckKeyPairExistenceResponse

Rappresenta la risposta della *persistence logic* alla richiesta di controllo della presenza delle chiavi.

Descrizione degli attributi della struttura:

- **`err error`**: rappresenta l'errore dell'operazione, se questo si controllo ,altrimenti il campo viene popolato con `nil`.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCheckKeyPairExistenceResponse(err error) *CheckKeyPairExistenceResponse`**: è il costruttore dell'oggetto;
- **`GetError() error`**: restituisce l'errore memorizzato nella risposta.

3.4.7.1.8 - GetPemPrivateKeyResponse

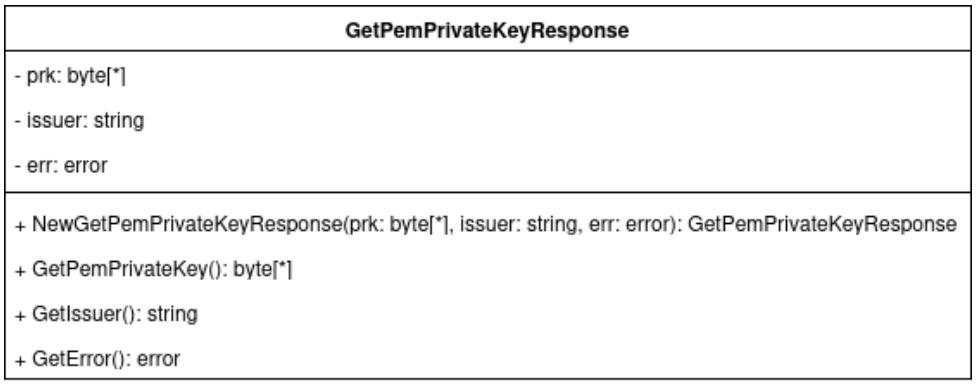

Figura 49: Authenticator - GetPemPrivateKeyResponse

Rappresenta la risposta della *persistence logic* alla richiesta di ottenimento della chiave privata.

Descrizione degli attributi della struttura:

- **`prk *[]byte`**: rappresenta la chiave privata, in formato Pem e memorizzata in byte;
- **`issuer string`**: rappresenta l'*issuer*, ovvero l'uuid generato al momento della creazione delle chiavi;
- **`err error`**: rappresenta l'errore dell'operazione, altrimenti il campo viene popolato con `nil`.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetPemPrivateKeyResponse(prk *[]byte, issuer string, err error) *GetPemPrivateKeyResponse`**: rappresenta il costruttore della risposta;
- **`GetIssuer() string`**: permette di ottenere l'*issuer* memorizzato nella risposta;
- **`GetPemPrivateKey() *[]byte`**: permette di ottenere la chiave privata memorizzata nella risposta;
- **`GetError() error`**: permette di ottenere l'errore memorizzato nella risposta.

3.4.7.1.9 - GetPemPublicKeyResponse

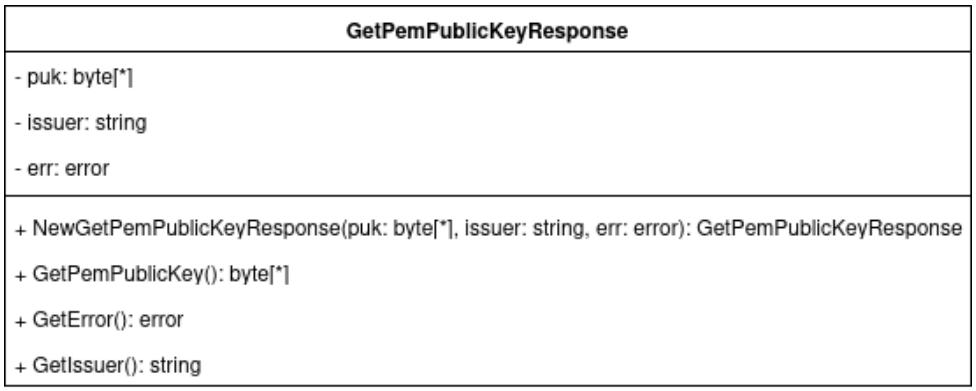

Figura 50: Authenticator - GetPemPublicKeyResponse

Rappresenta la risposta della *persistence logic* alla richiesta di ottenimento della chiave pubblica.

Descrizione degli attributi della struttura:

- **`puk *[]byte`**: rappresenta la chiave pubblica, in formato Pem e memorizzata in byte;
- **`issuer string`**: rappresenta l'*issuer*, ovvero l'uuid generato al momento della creazione delle chiavi;
- **`err error`**: rappresenta l'errore dell'operazione, altrimenti il campo viene popolato con `nil`.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetPemPublicKeyResponse(prk *[]byte, issuer string, err error) *GetPemPublicKeyResponse`**: rappresenta il costruttore della risposta;
- **`GetIssuer() string`**: permette di ottenere l'*issuer* memorizzato nella risposta;
- **`GetPemPublicKey() *[]byte`**: permette di ottenere la chiave pubblica memorizzata nella risposta;
- **`GetError() error`**: permette di ottenere l'errore memorizzato nella risposta.

3.4.7.1.10 - GetTokenResponse

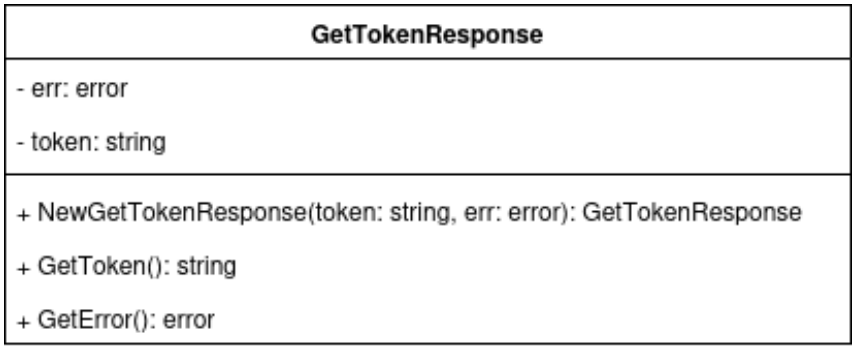

Figura 51: Authenticator - GetTokenResponse

Rappresenta la risposta alla richiesta di ottenimento Token.

Descrizione degli attributi della struttura:

- **`token string`**: rappresenta il Token generato e firmato, se presente;
- **`err error`**: rappresenta l'errore verificato durante l'elaborazione della richiesta, se presente, altrimenti è `nil`;

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetTokenResponse(Token string, err error) *GetTokenResponse`**: rappresenta il costruttore della risposta;
- **`GetToken() string`**: permette di ottenere il Token firmato memorizzato nella risposta;
- **`GetError() error`**: permette di ottenere l'errore memorizzato nella risposta.

3.4.7.1.11 - PublishPublicKeyResponse

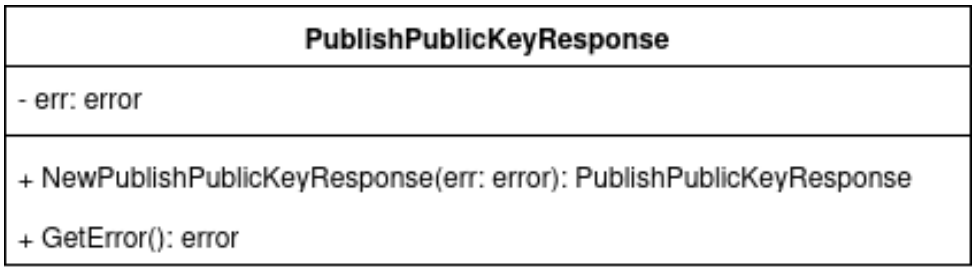

Figura 52: Authenticator - PublishPublicKeyResponse

Rappresenta la risposta alla richiesta di pubblicazione della chiave pubblica.

Descrizione degli attributi della struttura:

- **`err error`**: rappresenta l'errore dell'operazione, altrimenti il campo viene popolato con `nil`.

Descrizione dei metodi invocabili dalla struttura:

- **`NewPublishPublicKeyResponse(err error) *PublishPublicKeyResponse`**: è il costruttore dell'oggetto;
- **`GetError() error`**: restituisce l'errore memorizzato nella risposta.

3.4.7.1.12 - StorePemKeyPairResponse

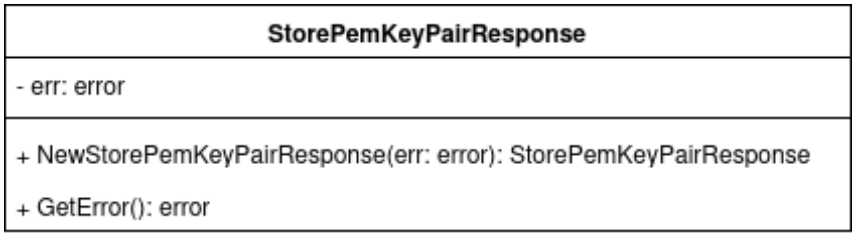

Figura 53: Authenticator - StorePemKeyPairResponse

Rappresenta la risposta alla richiesta di memorizzazione delle chiavi.

Descrizione degli attributi della struttura:

- **`err error`**: rappresenta l'errore dell'operazione, altrimenti il campo viene popolato con `nil`.

Descrizione dei metodi invocabili dalla struttura:

- **`NewStorePemKeyPairResponse(err error) *StorePemKeyPairResponse`**: è il costruttore dell'oggetto;
- **`GetError() error`**: restituisce l'errore memorizzato nella risposta.

3.4.7.1.13 - PemPrivateKey

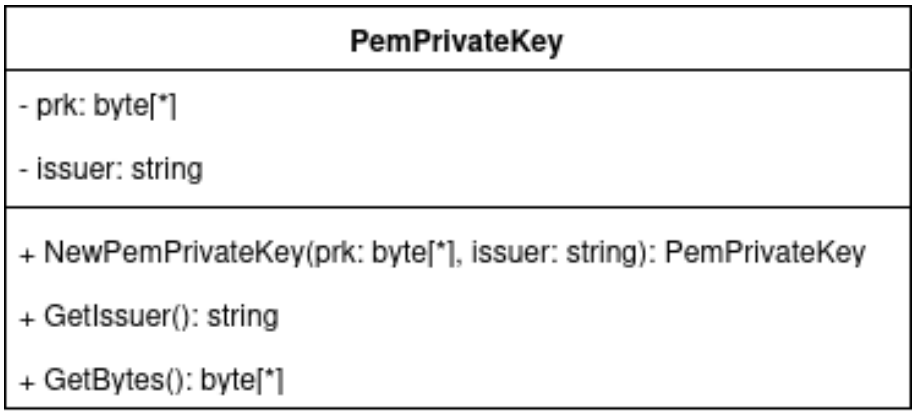

Figura 54: Authenticator - PemPrivateKey

Oggetto utilizzato per memorizzare la chiave privata in formato Pem e l'*issuer* della stessa.

Descrizione degli attributi della struttura:

- **`prk *[]byte`**: rappresenta la chiave privata, in formato Pem e memorizzata in byte;
- **`issuer string`**: rappresenta l'*issuer*, ovvero l'uuid generato al momento della creazione delle chiavi.

Descrizione dei metodi invocabili dalla struttura:

- **`NewPemPrivateKey(prk *[]byte, issuer string) *PemPrivateKey`**: è il costruttore dell'oggetto;
- **`GetIssuer() string`**: permette di ottenere l'*issuer* memorizzato;
- **`GetBytes() []byte`**: permette di ottenere una copia dei *bytes* memorizzati della chiave privata;

3.4.7.1.14 - PemPublicKey

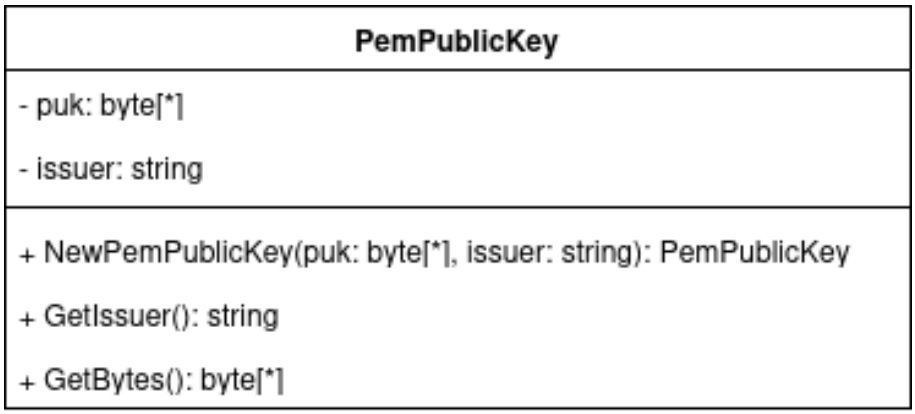

Figura 55: Authenticator - PemPublicKey

Oggetto utilizzato per memorizzare la chiave pubblica in formato Pem e l'*issuer* della stessa.

Descrizione degli attributi della struttura:

- **`prk *[]byte`**: rappresenta la chiave pubblica, in formato Pem e memorizzata in byte;
- **`issuer string`**: rappresenta l'*issuer*, ovvero l'uuid generato al momento della creazione delle chiavi.

Descrizione dei metodi invocabili dalla struttura:

- **`NewPemPublicKey(puk *[]byte, issuer string) *PemPublicKey`**: è il costruttore dell'oggetto;
- **`GetIssuer() string`**: permette di ottenere l'*issuer* memorizzato;
- **`GetBytes() []byte`**: permette di ottenere una copia dei *bytes* memorizzati della chiave pubblica;

3.4.7.2 - IAuthPersistence

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* del microservizio *Authenticator*.

Descrizione dei metodi dell'interfaccia:

- **`StorePemKeyPair(prk []byte, puk []byte, emit string) error`**: il metodo deve dare la possibilità di memorizzare una coppia di chiavi ECDSA in formato Pem e l'identificatore (uuid) che verrà poi associato come *issuer* delle chiavi generate;
- **`GetPemPublicKey() (PemPublicKey, error)`**: il metodo deve dare la possibilità di ottenere l'oggetto `PemPublicKey` memorizzato. Se non presente, error deve essere diverso da `nil`;
- **`GetPemPrivateKey() (PemPrivateKey, error)`**: il metodo deve dare la possibilità di ottenere l'oggetto `PemPrivateKey` memorizzato. Se non presente, error deve essere diverso da `nil`;
- **`CheckKeyPairExistence() error`**: il metodo deve restituire un errore se non è memorizzata una coppia di chiavi, altrimenti deve restituire `nil`.

3.4.7.3 - AuthRepository

Questa struttura implementa l'interfaccia **IAuthPersistence**, vedi la Sezione 3.4.7.2.

Descrizione degli attributi della struttura:

- **`prk *PemPrivateKey`**: un puntatore all'oggetto `PemPrivateKey` che contiene la chiave privata, se questa è stata memorizzata, altrimenti è `nil`;
- **`prk *PemPublicKey`**: un puntatore all'oggetto `PemPublicKey` che contiene la chiave pubblica, se questa è stata memorizzata, altrimenti è `nil`;
- **`issuer string`**: è l'identificatore (uuid) di chi ha emesso le chiavi al momento della loro memorizzazione. Se le chiavi non sono ancora state memorizzate, la stringa è vuota;
- **`mutex sync.Mutex`**: variabile utilizzata per il corretto funzionamento di alcuni metodi. Si rimanda alla documentazione del linguaggio Go[G] per ulteriori informazioni.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthRepo() *AuthRepository`**: costruttore della struttura. Non prende alcun parametro e inizializza i puntatori a `nil` e la stringa come stringa vuota;
- **`checkKeyPair(prk *[]byte, puk *[]byte) bool`**: metodo che assolve al compito di controllo validità delle chiavi passate come parametro in formato Pem. Se le chiavi non sono in formato Pem e/o non sono chiavi ECDSA allora viene restituito `false`, altrimenti viene restituito `true`;
- **`StorePemKeyPair(prk []byte, puk []byte, emit string) error`**: il metodo controlla la validità delle chiavi passate utilizzando la funzione `checkKeyPair` e, se il controllo è positivo, le memorizza assieme all'*issuer*;
- **`GetPemPublicKey() (PemPublicKey, error)`**: permette di ottenere l'oggetto `PemPublicKey`, se presente, altrimenti viene restituito un oggetto vuoto e un errore;

- **`GetPemPrivateKey() (PemPrivateKey, error)`**: permette di ottenere l'oggetto `PemPrivateKey`, se presente, altrimenti viene restituito un oggetto vuoto e un errore;
- **`CheckKeyPairExistence() error`**: restituisce un errore se non vi è una coppia di chiavi memorizzata, altrimenti `nil`.

3.4.7.4 - ICheckKeyPairExistance

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di voler verificare l'esistenza di una coppia di chiavi.

Descrizione dei metodi dell'interfaccia:

`CheckKeyPairExistance(cmd *servicecmd.CheckPemKeyPairExistenceCmd) *serviceresponse.CheckKeyPairExistenceResponse`: il metodo deve offrire la possibilità di richiedere il controllo dell'esistenza di una coppia di chiavi memorizzata e ricevere adeguata risposta.

3.4.7.5 - IGetPemPrivateKeyPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere la chiave privata, se memorizzata.

Descrizione dei metodi dell'interfaccia:

- **`GetPemPrivateKey(cmd *servicecmd.GetPemPrivateKeyCmd) *serviceresponse.GetPemPrivateKeyResponse`**: il metodo deve offrire la possibilità di richiedere la chiave privata memorizzata e ottenere un'adeguata risposta;

3.4.7.6 - IGetPemPublicKeyPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere la chiave pubblica, se memorizzata.

Descrizione dei metodi dell'interfaccia:

- **`GetPemPublicKey(cmd *servicecmd.GetPemPublicKeyCmd) *serviceresponse.GetPemPublicKeyResponse`**: il metodo deve offrire la possibilità di richiedere la chiave pubblica memorizzata e ottenere un'adeguata risposta;

3.4.7.7 - IStorePemKeyPair

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di memorizzare una coppia di chiavi e l'*issuer* associato.

Descrizione dei metodi dell'interfaccia:

- **`StorePemKeyPair(cmd *servicecmd.StorePemKeyPairCmd) *serviceresponse.StorePemKeyPairResponse`**: il metodo deve permettere di richiedere la memorizzazione di una coppia di chiavi e l'*issuer* associato e ricevere una risposta in merito all'esito dell'operazione

3.4.7.8 - AuthAdapter

Adapter che mette in comunicazione la *business logic* con la *persistence logic*. Implementa le seguenti interfacce (porte):

- **`ICheckKeyPairExistance`**, Sezione 3.4.7.4;
- **`IGetPemPrivateKeyPort`**, Sezione 3.4.7.5;
- **`IGetPemPublicKeyPort`**, Sezione 3.4.7.6;
- **`IStorePemKeyPair`**, Sezione 3.4.7.7.

Descrizione degli attributi della struttura:

- **`repo persistence.IAuthPersistence`**: l'*Adapter* possiede un attributo alla struttura rappresentante la *persistence logic* di Authenticator. Per ulteriori informazioni si veda la Sezione 3.4.7.2.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthAdapter(repo persistence.IAuthPersistence) *AuthAdapter`**: costruttore dell'oggetto. Prende l'oggetto che implementa la *persistence logic* come parametro;
- **`StorePemKeyPair(cmd *servicecmd.StorePemKeyPairCmd) *serviceresponse.StorePemKeyPairResponse`**: converte il *Command* per la memorizzazione della coppia di chiavi in formato Pem e l'issuer associato in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;
- **`GetPemPrivateKey(cmd *servicecmd.GetPemPrivateKeyCmd) *serviceresponse.GetPemPrivateKeyResponse`**: converte il *Command* per l'ottenimento della chiave privata e del suo issuer in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;
- **`GetPemPublicKey(cmd *servicecmd.GetPemPublicKeyCmd) *serviceresponse.GetPemPublicKeyResponse`**: converte il *Command* per l'ottenimento della chiave pubblica e del suo issuer in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;
- **`CheckKeyPairExistance(cmd *servicecmd.CheckPemKeyPairExistenceCmd) *serviceresponse.CheckKeyPairExistenceResponse`**: converte il *Command* per il controllo della presenza di una coppia di chiavi memorizzata in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata.

3.4.7.9 - IPublishPort

Rappresenta la porta che consente alla *business logic* di comunicare al *Publisher* la volontà di pubblicare la propria chiave pubblica.

Descrizione dei metodi dell'interfaccia:

- **`PublishKey(cmd *servicecmd.PublishPublicKeyCmd) *serviceresponse.PublishPublicKeyResponse`**: il metodo deve dare la possibilità di richiedere la pubblicazione della chiave pubblica e ricevere una risposta sull'esito dell'operazione.

3.4.7.10 - PublishAdapter

Rappresenta l'*Adapter* atto a convertire il *Command* per la richiesta di pubblicazione della chiave pubblica al *Publisher*.

Implementa l'interfaccia IPublishPort, descritta alla Sezione 3.4.7.9.

Descrizione degli attributi della struttura:

- **`pb publisher.IAuthPublisher`**: istanza di una struttura che rappresenta il *Publisher*, vedi Sezione 3.4.7.11 per maggiori informazioni.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthPublisherAdapter(pb publisher.IAuthPublisher) *AuthPublisherAdapter`** : costruttore dell'*Adapter*. Prende un oggetto che implementa l'interfaccia per i *Publisher* come parametro;
- **`PublishKey(cmd *servicecmd.PublishPublicKeyCmd) *serviceresponse.PublishPublicKeyResponse`**: prende come parametro il *Command* per la pubblicazione della chiave pubblica, ne prende la chiave pubblica e l'issuer e li passa al *Publisher* per la pubblicazione.

3.4.7.11 - IAuthPublisher

Interfaccia che gli oggetti rappresentanti un *Publisher* per le chiavi pubbliche devono soddisfare.

Descrizione dei metodi dell'interfaccia:

- **`PublishKey(puk crypto.PublicKey, issuer string) error`**: il metodo deve permettere la pubblicazione della chiave pubblica e dell'*issuer* associato come parametro, quindi deve restituire un errore se la richiesta non è stata completata, o `nil` altrimenti.

3.4.7.12 - AuthPublisher

Un *Publisher* che permette di pubblicare su un JetStream NATS[G] chiave pubblica e issuer associato.

Implementa l'interfaccia **IAuthPublisher**: per maggiori informazioni vedere la Sezione 3.4.7.11.

Descrizione degli attributi della struttura:

- **`mb *broker.NatsMessageBroker`**: un'istanza di un *message broker* di NATS[G], necessario per la pubblicazione delle chiavi e dell'*issuer*;

Descrizione dei metodi invocabili dalla struttura:

- **`NewPublisher(mb *broker.NatsMessageBroker) *AuthPublisher`**: costruttore della struttura, prende un *message broker* di NATS[G] come parametro;
- **`PublishKey(puk crypto.PublicKey, issuer string) error`**: permette di pubblicare la chiave e l'issuer passati come parametro in un apposito JetStream di NATS[G], ritornando un errore se l'operazione non va a buon fine.

3.4.7.13 - UserData

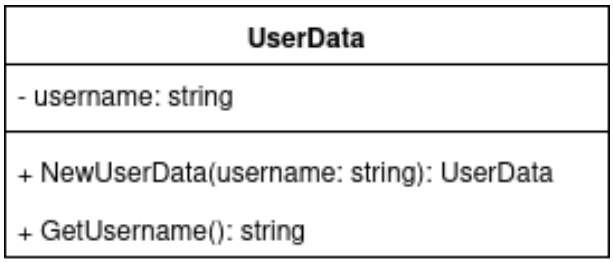

Figura 56: Authenticator - UserData

Oggetto utilizzato da IAuthenticateUserStrategy, vedi Sezione 3.4.7.14.

Descrizione degli attributi della struttura:

- **`username string`**: nome utente da verificare.

Descrizione dei metodi invocabili dalla struttura:

- **`NewUserData(username string) *UserData`**: costruttore della struttura;
- **`GetUsername() string`**: fornisce il nome utente memorizzato nell'istanza della struttura che lo invoca.

3.4.7.14 - IAuthenticateUserStrategy

Interfaccia che le strutture che implementano l'algoritmo per valutare i dati di autenticazione forniti devono soddisfare.

Descrizione dei metodi dell'interfaccia:

- **`Authenticate(us serviceobject.UserData) (string, error)`**: il metodo deve ospitare l'algoritmo che controlla i dati di autenticazione. Deve restituire uno stringa con il ruolo e `nil` se il controllo ha esito positivo, una stringa vuota e un errore altrimenti.

3.4.7.15 - SimpleAuthAlg

Implementazione di un semplice algoritmo per verificare i dati di autenticazione: con lo scopo di soddisfare l'MVP, questo algoritmo controlla se lo *username* è tra quelli prestabiliti.

Descrizione degli attributi della struttura:

- **`usernameRoles map[string]string`**: mappa contenente gli username autorizzati e i rispettivi ruoli;
- **`mutex sync.Mutex`**, variabile utilizzata per il corretto funzionamento di alcuni metodi. Si rimanda alla documentazione del linguaggio Go[G] per ulteriori informazioni.

Descrizione dei metodi invocabili dalla struttura:

- **`Authenticate(us serviceobject.UserData) (string, error)`**: il metodo controlla lo username. Se il controllo va a buon fine viene ritornato il ruolo e `nil`, altrimenti viene ritornata una stringa vuota e un errore.

3.4.7.16 - IGetTokenUseCase

Interfaccia che permette all'*application logic* di comunicare alla *business logic* la necessità di ottenere un Token.

Descrizione dei metodi dell'interfaccia:

- **`GetToken(cmd *servicecmd.GetTokenCmd) *serviceresponse.GetTokenResponse`**: il metodo deve permettere, dato un *Command* per la richiesta di un Token passato come parametro, di ottenere, sotto forma di risposta adeguata, il suddetto Token.

3.4.7.17 - AuthService

Oggetto rappresentante la *business logic* di **Authenticator**.

Implementa l'interfaccia (*Use Case*) **IGetTokenUseCase**, per maggiori informazioni vedere la Sezione 3.4.7.16.

Descrizione degli attributi della struttura:

- **`checkKeyPairExistancePort serviceportout.ICheckKeyPairExistance`**, si veda la Sezione 3.4.7.4;
- **`getPemPrivateKeyPort serviceportout.IGetPemPrivateKeyPort`**, si veda la Sezione 3.4.7.5;
- **`getPemPublicKeyPort serviceportout.IGetPemPublicKeyPort`**, si veda la Sezione 3.4.7.6;

- **`storePemKeyPairPort serviceportout.IStorePemKeyPair`**, si veda la Sezione 3.4.7.7;
- **`publishPort serviceportout.IPublishPort`**, si veda la Sezione 3.4.7.9;
- **`authenticatorStrategy serviceauthenticator.IAuthenticateUserStrategy`**, si veda la Sezione 3.4.7.14;
- **`mutex sync.Mutex`**, variabile utilizzata per il corretto funzionamento di alcuni metodi. Si rimanda alla documentazione del linguaggio Go[G] per ulteriori informazioni.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthService(p AuthServiceParams) *AuthService`**: costruttore dell'oggetto. Le porte (*Use Case*) devono essere fornite come parametri al costruttore e, per farlo, si utilizza la struttura **`AuthServiceParams`**, struttura con i medesimi attributi di AuthService con l'istruzione `fx.in` per permettere al *framework* **fx** di fornire automaticamente le dipendenze necessarie;
- **`generatePemKey() (*[]byte, *[]byte, error)`**: genera una coppia di chiavi ECDSA e le converte in formato Pem, quindi le ritorna con `nil` come errore. Se la richiesta non va a buon fine vengono ritornati due puntatori a `nil` e un errore;
- **`storePemKeyPair(cmd *servicecmd.StorePemKeyPairCmd) error`**: gestisce la memorizzazione delle chiavi, quando generate;
- **`getPrivateKeyFromPem(prk *[]byte) (*ecdsa.PrivateKey, error)`**: si occupa di convertire una chiave privata passata in formato Pem in una chiave privata ECDSA a tutti gli effetti;
- **`generateToken(username string, role string) (string, error)`**: si occupa di generare e firmare un Token;
- **`GetToken(cmd *servicecmd.GetTokenCmd) *serviceresponse.GetTokenResponse`**: esegue un controllo delle credenziali e, se il controllo ha esito positivo, fa generare un Token.

3.4.7.18 - AuthController

È l'oggetto che gestisce l'*application logic*.

Descrizione degli attributi della struttura:

- **`tokenUseCase serviceportin.IGetTokenUseCase`**: vedi Sezione 3.4.7.16.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthController(tokenUseCase serviceportin.IGetTokenUseCase, mp MetricParams) *authController`**: costruttore dell'oggetto. Prende un oggetto che soddisfa `IGetTokenUseCase` e una struttura, dotata di comando `fx.In`, contenente quanto necessario per effettuare misurazioni, come parametro;
- **`checkGetTokenRequest(dto *common.AuthLoginRequest) error`**: controlla la correttezza della richiesta per ottenere un Token e restituisce un errore in caso di risultato negativo o `nil` altrimenti;
- **`NewTokenRequest(ctx context.Context, msg *nats.Msg) error`**: si occupa di gestire una richiesta di ottenimento Token e rispondere alla stessa con il Token o con una stringa vuota se la procedura non va a buon fine.

3.4.8 - Order

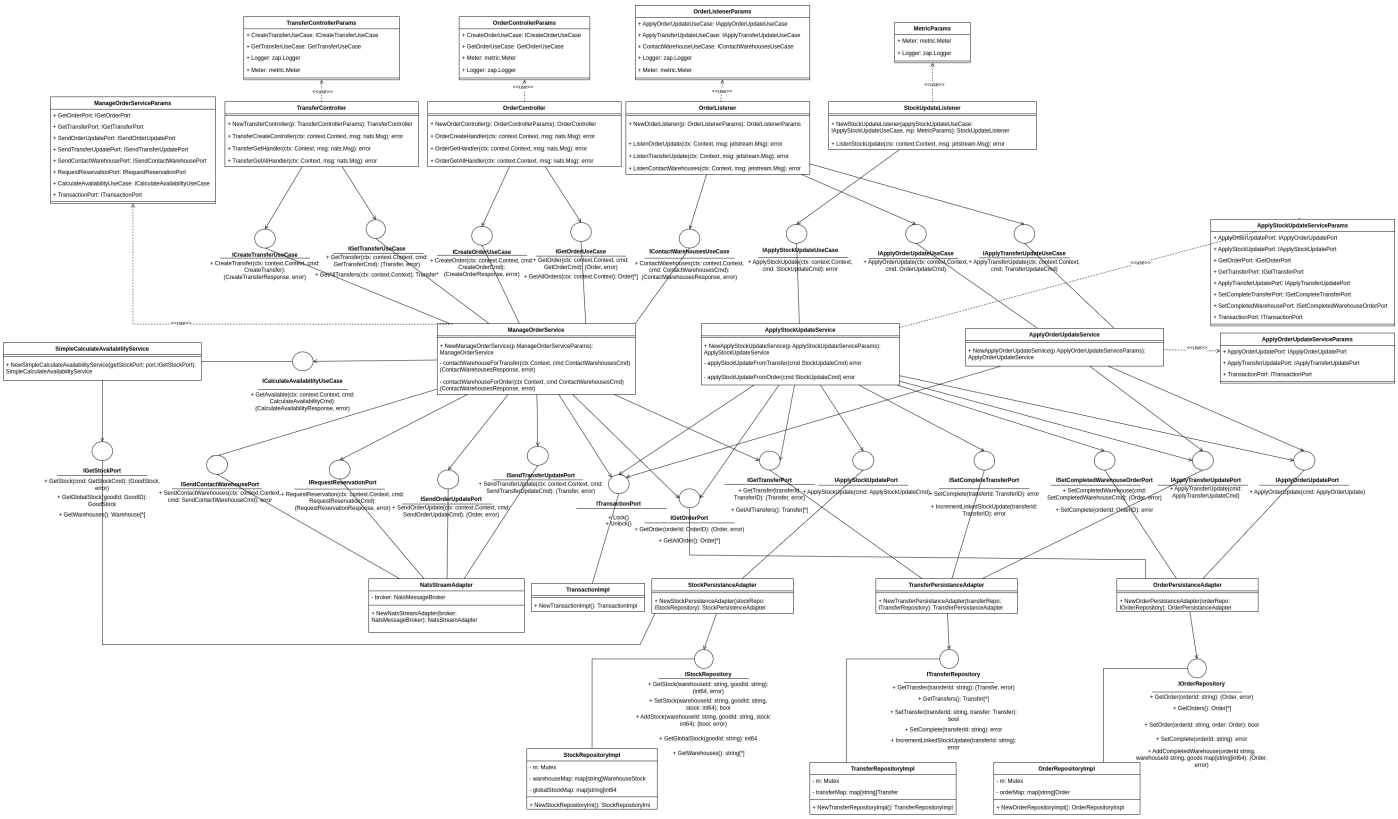

Figura 57: Struttura del Microservizio «Order»

Il microservizio **Order** viene utilizzato per realizzare gli ordini quando questi vengono confermati, andando a verificare la disponibilità di una merce e organizzando da quali magazzini prendere la merce per completare l'ordine.

È sempre di questo microservizio il compito di assolvere ai trasferimenti, particolari tipi di ordini il cui destinatario è un altro magazzino.

In particolare, è formato da tre sotto-componenti principali:

- I **Controller** e **Listener**, che rappresentano l'*application logic*
- I **Service**, che rappresentano la *business logic*;
- I **Repository**, che rappresentano la *persistence logic*.

Gli oggetti utilizzati per implementare queste componenti saranno ora esposti.

3.4.8.1 - Oggetti comuni del microservizio

3.4.8.1.1 - OrderWarehouseUsed (Repo)

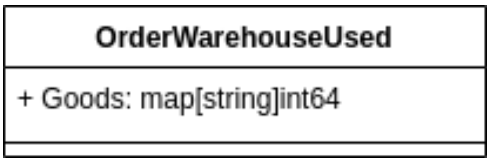

Figura 58: Order - OrderWarehouseUsed (Repo)

Rappresenta un magazzino utilizzato per soddisfare un ordine[G]. Questo oggetto è utilizzato nella *persistence logic* per memorizzare le informazioni sui magazzini coinvolti nella gestione degli ordini.

Descrizione degli attributi della struttura:

- **`Goods map[string]int64`**: rappresenta una mappa che associa l'identificativo della merce (**string**) alla quantità utilizzata (**int64**).

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.2 - OrderUpdateGood (Repo)

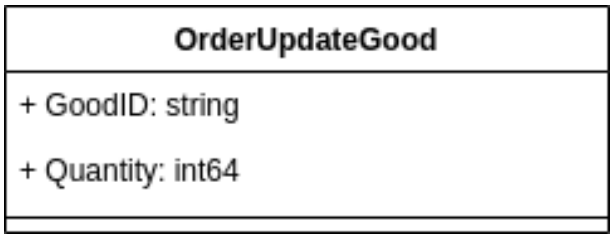

Figura 59: Order - OrderUpdateGood (Repo)

Rappresenta una merce coinvolta in un aggiornamento di un ordine[G]. Questo oggetto è utilizzato nella *persistence logic* per memorizzare i dettagli delle merci associate agli ordini.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nell'ordine;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.3 - Order (Repo)

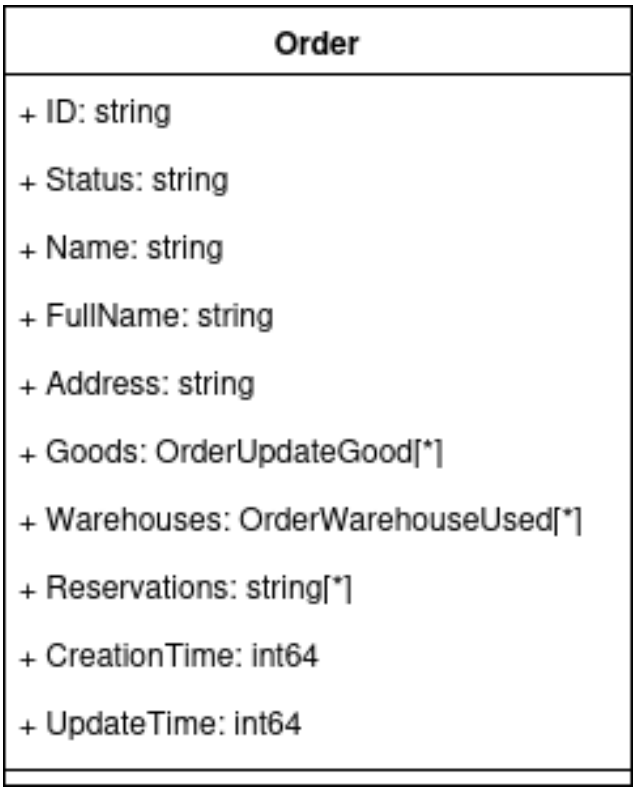

Figura 60: Order - Order (Repo)

Rappresenta un ordine[G] nel sistema. Questo oggetto è utilizzato nella *persistence logic* per memorizzare e gestire le informazioni relative agli ordini, inclusi i dettagli delle merci, i magazzini coinvolti e le prenotazioni associate.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine;
- **`Status string`**: rappresenta lo stato dell'ordine;
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []OrderUpdateGood`**: rappresenta una lista di oggetti `OrderUpdateGood` che contengono le informazioni sulle merci coinvolte nell'ordine;
- **`Warehouses []OrderWarehouseUsed`**: rappresenta una lista di oggetti `OrderWarehouseUsed` che contengono le informazioni sui magazzini utilizzati per l'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.4 - WarehouseStock (Repo)

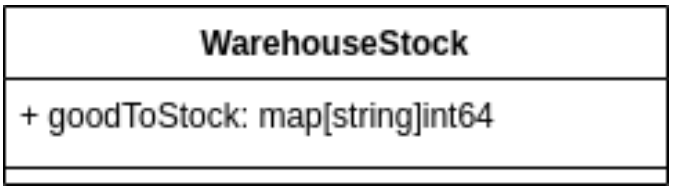

Figura 61: Order - WarehouseStock (Repo)

Descrizione degli attributi della struttura:

- **`goodToStock map[string]int64`**:

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.5 - TransferUpdateGood (Repo)

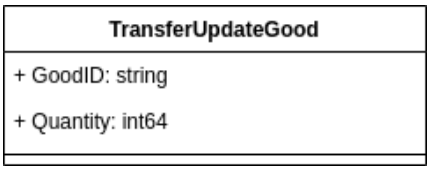

Figura 62: Order - TransferUpdateGood (Repo)

Rappresenta una merce coinvolta in un aggiornamento di un trasferimento[G]. Questa struttura viene utilizzata per memorizzare i dettagli delle merci associate a un trasferimento[G], come l'identificativo della merce e la quantità coinvolta.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nel trasferimento[G]
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nel trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.6 - Transfer (Repo)

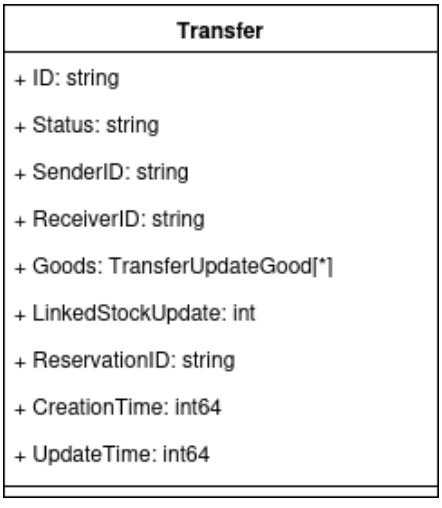

Figura 63: Order - Transfer (Repo)

Rappresenta un trasferimento[G] nel sistema. Questo oggetto è utilizzato nella *persistence logic* per memorizzare e gestire le informazioni relative ai trasferimenti, inclusi i dettagli delle merci, i magazzini coinvolti e le prenotazioni associate.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del trasferimento[G]
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G] (es. «Created», «Filled»);
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G]
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`LinkedStockUpdate int`**: rappresenta il numero di aggiornamenti dello stock[G] associati al trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G]
- **`Goods []TransferUpdateGood`**: rappresenta una lista di oggetti `TransferUpdateGood` che contengono le informazioni sulle merci coinvolte nel trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.7 - TransferID

Rappresenta un identificativo univoco per un trasferimento[G].

Descrizione degli attributi della struttura:

- **`string`**: rappresenta l'identificativo univoco del trasferimento[G].

3.4.8.1.8 - Transfer

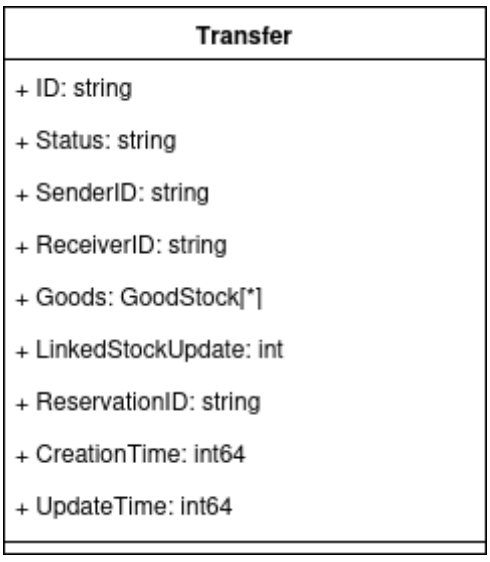

Figura 64: Order - Transfer

Rappresenta un trasferimento[G] nel sistema. Questo oggetto è utilizzato nella *business logic* per gestire le informazioni relative ai trasferimenti, inclusi i dettagli delle merci, i magazzini coinvolti e le prenotazioni associate.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del trasferimento[G]
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G] (es. «Created», «Filled»);
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G]
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`LinkedStockUpdate int`**: rappresenta il numero di aggiornamenti dello stock[G] associati al trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G]
- **`Goods []GoodStock`**: rappresenta una lista di oggetti `GoodStock` che contengono le informazioni sulle merci coinvolte nel trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.9 - OrderID

Rappresenta un identificativo univoco per un ordine[G].

Descrizione degli attributi della struttura:

- **`string`**: rappresenta l'identificativo univoco dell'ordine.

3.4.8.1.10 - Order

Figura 65: Order - Order

Rappresenta un ordine[G] nel sistema.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine;

- **`Status string`**: rappresenta lo stato dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []GoodStock`**: rappresenta una lista di oggetti `GoodStock` che contengono le informazioni sulle merci incluse nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine;
- **`Warehouses []OrderWarehouseUsed`**: rappresenta una lista di oggetti `OrderWarehouseUsed` che contengono le informazioni sui magazzini utilizzati per l'ordine.

Descrizione dei metodi invocabili dalla struttura:

- **`IsCompleted() bool`**: verifica[G] se l'ordine è stato completato. Restituisce `true` se tutte le merci richieste sono state soddisfatte dai magazzini, altrimenti `false`.

3.4.8.1.11 - OrderWarehouseUsed

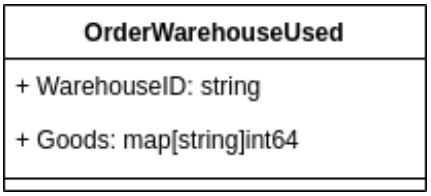

Figura 66: Order - OrderWarehouseUsed

Rappresenta un magazzino utilizzato per soddisfare un ordine[G].

Descrizione degli attributi della struttura:

- **`WarehouseID string`**: rappresenta l'identificativo del magazzino utilizzato;
- **`Goods map[string]int64`**: rappresenta una mappa che associa l'identificativo della merce (**string**) alla quantità utilizzata (**int64**).

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.12 - WarehouseID

Rappresenta un identificativo univoco per un magazzino.

Descrizione degli attributi della struttura:

- **`string`**: rappresenta l'identificativo univoco del magazzino.

3.4.8.1.13 - Warehouse

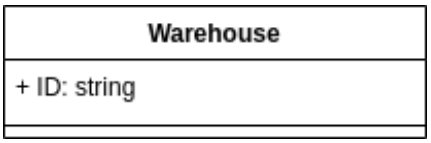

Figura 67: Order - Warehouse

Rappresenta un magazzino nel sistema.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del magazzino.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.14 - CreateTransferCmd

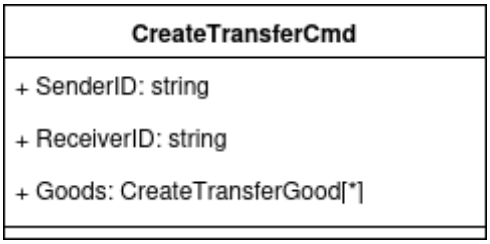

Figura 68: Order - CreateTransferCmd

Rappresenta il comando utilizzato per creare un nuovo trasferimento[G] nel sistema.

Descrizione degli attributi della struttura:

- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []CreateTransferGood`**: rappresenta una lista di oggetti `CreateTransferGood` che contengono le informazioni sulle merci incluse nel trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.15 - CreateTransferGood

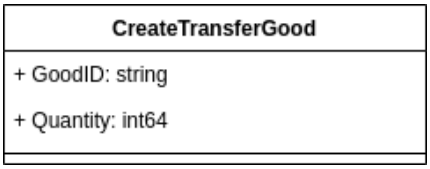

Figura 69: Order - CreateTransferGood

Rappresenta una merce inclusa in un trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce inclusa nel trasferimento[G]
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nel trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.16 - CreateTransferResponse

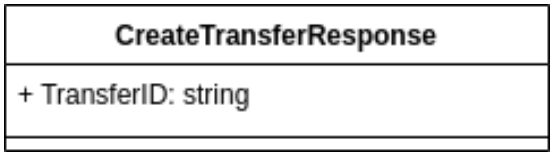

Figura 70: Order - CreateTransferResponse

Rappresenta la risposta alla richiesta di creazione di un nuovo trasferimento[G].

Descrizione degli attributi della struttura:

- **`TransferID string`**: rappresenta l'identificativo univoco del trasferimento[G] creato.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.17 - ApplyOrderUpdateCmd

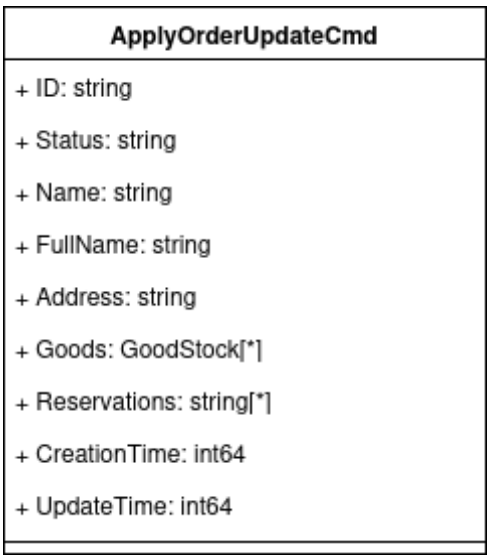

Figura 71: Order - ApplyOrderUpdateCmd

Rappresenta il comando utilizzato per applicare un aggiornamento di un ordine[G].

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine da aggiornare;
- **`Status string`**: rappresenta lo stato dell'ordine da aggiornare;
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []model.GoodStock`**: rappresenta una lista di oggetti `GoodStock` che contengono le informazioni sulle merci coinvolte nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine.

3.4.8.1.18 - ApplyTransferUpdateCmd

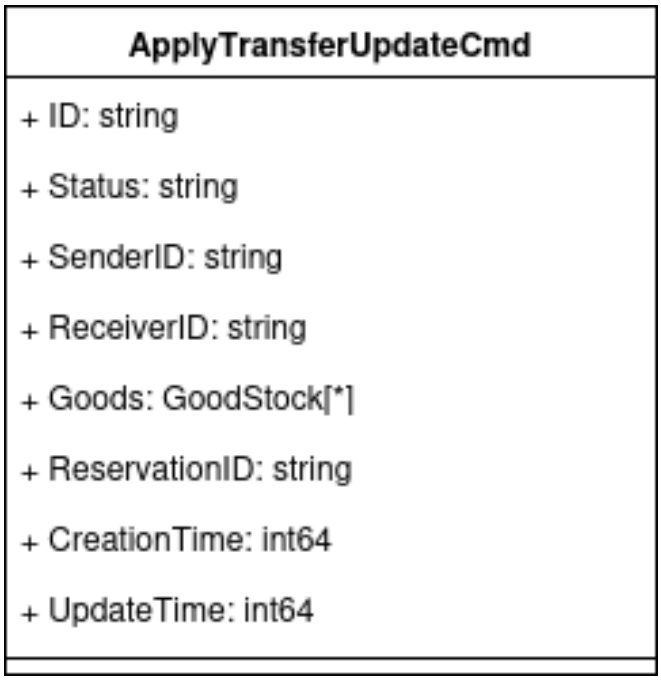

Figura 72: Order - ApplyTransferUpdateCmd

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'aggiornamento del trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G] aggiornato (es. «Created», «Filled», «Completed»);
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []model.GoodStock`**: rappresenta una lista di oggetti `GoodStock` che contengono le informazioni sulle merci coinvolte nel trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G]
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G]
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.19 - SetCompletedWarehouseCmd

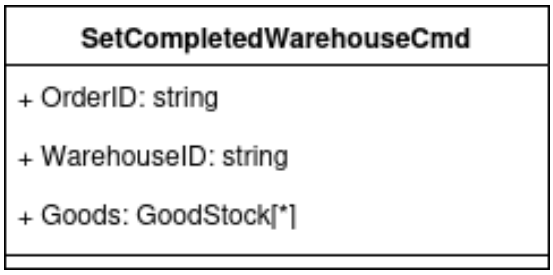

Figura 73: Order - SetCompletedWarehouseCmd

Rappresenta il comando utilizzato per segnalare il completamento di un ordine[G] da parte di un magazzino.

Descrizione degli attributi della struttura:

- **`OrderID string`**: rappresenta l'identificativo univoco dell'ordine in questione;
- **`WarehouseID string`**: rappresenta l'identificativo del magazzino che ha completato l'ordine;
- **`Goods []model.GoodStock`**: rappresenta una lista di oggetti `GoodStock` che contengono le informazioni sulle merci coinvolte nell'ordine completato.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.20 - TransferUpdateCmd

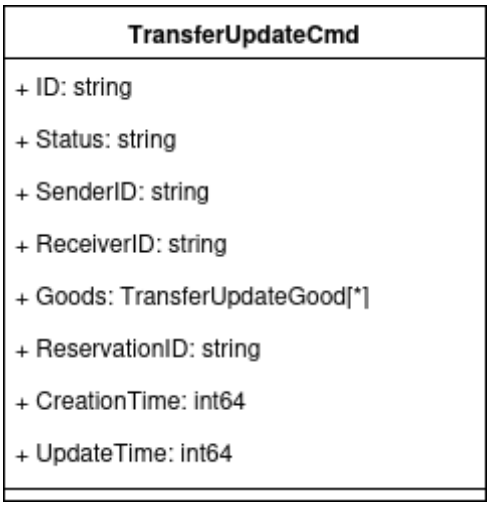

Figura 74: Order - TransferUpdateCmd

Rappresenta il comando utilizzato per aggiornare le informazioni di un trasferimento[G] nel sistema.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del trasferimento[G] aggiornato;
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []TransferUpdateGood`**: rappresenta una lista di oggetti `TransferUpdateGood` che contengono le informazioni sulle merci coinvolte nel trasferimento[G]

- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G] aggiornato;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G].

3.4.8.1.21 - TransferUpdateGood

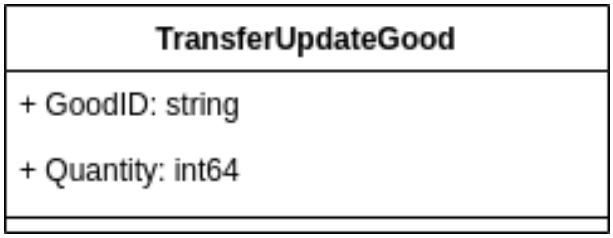

Figura 75: Order - TransferUpdateGood

Rappresenta una merce coinvolta in un aggiornamento di un trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nel trasferimento[G]
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nel trasferimento[G].

3.4.8.1.22 - OrderUpdateCmd

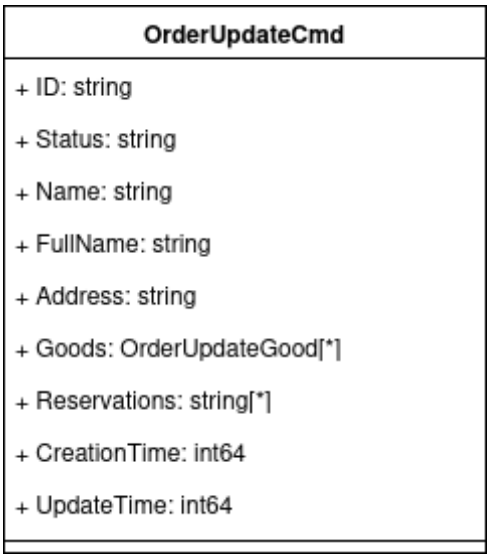

Figura 76: Order - OrderUpdateCmd

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine aggiornato;
- **`Goods []OrderUpdateGood`**: rappresenta una lista di oggetti `OrderUpdateGood` che contengono le informazioni sulle merci coinvolte nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine;
- **`Status string`**: rappresenta lo stato dell'ordine aggiornato;
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;

- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.23 - OrderUpdateGood

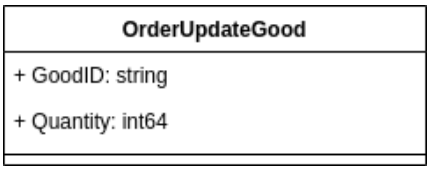

Figura 77: Order - OrderUpdateGood

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nell'ordine;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.24 - GoodID

Rappresenta un identificativo univoco per una merce.

Descrizione del tipo:

- **`string`**: il tipo `GoodID` è un alias per il tipo `string`. Viene utilizzato per identificare in modo univoco una merce all'interno del sistema.

3.4.8.1.25 - GoodStock

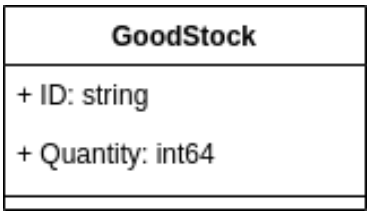

Figura 78: Order - GoodStock

Rappresenta lo stock[G] di una merce, ovvero la sua quantità.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'id della merce in questione;
- **`Quantity int64`**: rappresenta la quantità della merce.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.26 - StockUpdateGood

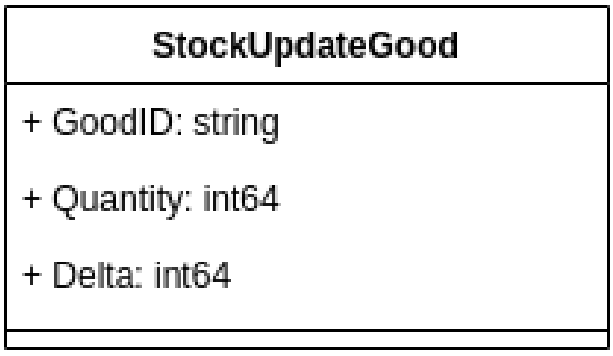

Figura 79: Order - StockUpdateGood

Rappresenta una merce coinvolta in un aggiornamento dello stock[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce coinvolta nell'aggiornamento dello stock[G]
- **`Quantity int64`**: rappresenta la quantità attuale della merce;
- **`Delta int64`**: rappresenta la differenza di quantità della merce rispetto all'ultimo stato.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.27 - StockUpdateType

Rappresenta il tipo di aggiornamento dello stock[G].

Descrizione dei valori possibili:

- **`add`**: per aggiungere stock[G]
- **`remove`**: per rimuovere stock[G]
- **`order`**: per aggiornamenti legati a ordini;
- **`transfer`**: per aggiornamenti legati a trasferimenti.

3.4.8.1.28 - StockUpdateCmd

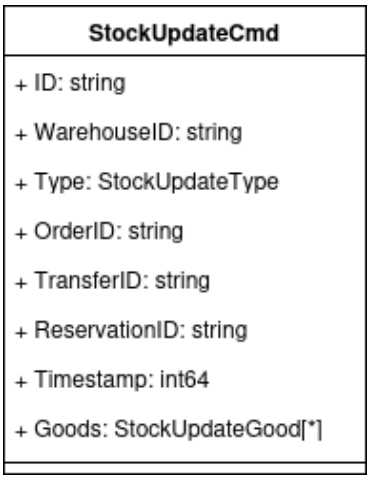

Figura 80: Order - StockUpdateCmd

Rappresenta il comando per aggiornare lo stock[G].

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del comando di aggiornamento dello stock[G]
- **`WarehouseID string`**: rappresenta l'identificativo del magazzino coinvolto nell'aggiornamento dello stock[G]
- **`Type StockUpdateType`**: rappresenta il tipo di aggiornamento dello stock[G]. Può assumere i valori descritti alla Sezione 3.4.8.1.27;
- **`Goods []StockUpdateGood`**: rappresenta una lista di oggetti `StockUpdateGood` che contengono le informazioni sulle merci coinvolte nell'aggiornamento;
- **`OrderID string`**: rappresenta l'identificativo dell'ordine associato all'aggiornamento dello stock[G]
- **`TransferID string`**: rappresenta l'identificativo del trasferimento[G] associato all'aggiornamento dello stock[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata all'aggiornamento dello stock[G]
- **`Timestamp int64`**: rappresenta il *timestamp* dell'aggiornamento dello stock[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.29 - GetStockCmd

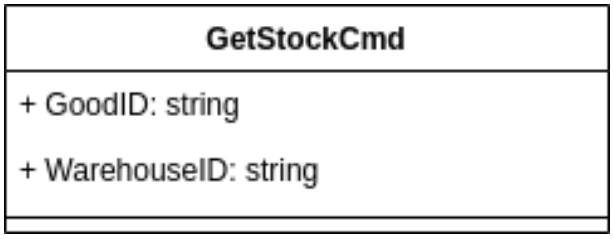

Figura 81: Order - GetStockCmd

Rappresenta il comando utilizzato per ottenere la quantità di una merce specifica in un determinato magazzino.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce di cui si vuole ottenere la quantità;
- **`WarehouseID string`**: rappresenta l'identificativo del magazzino in cui si vuole verificare la quantità della merce.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.30 - ContactWarehousesCmd

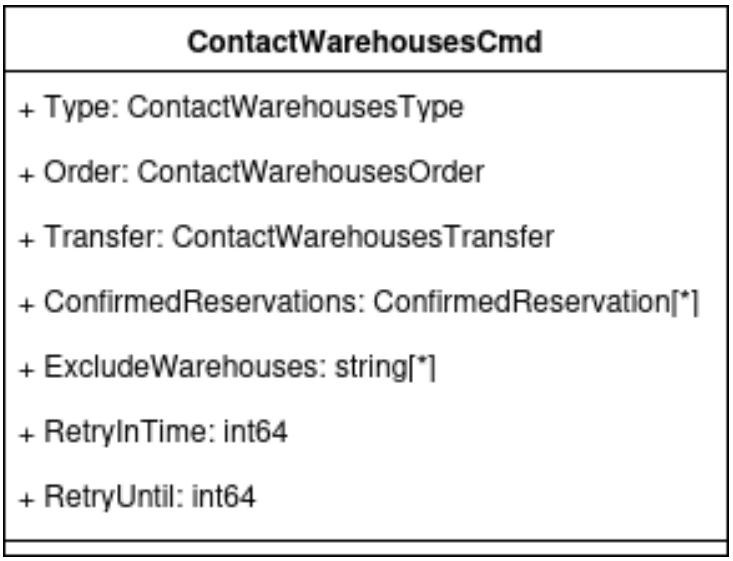

Figura 82: Order - ContactWarehousesCmd

Rappresenta il comando utilizzato per contattare i magazzini al fine di gestire ordini o trasferimenti.

Descrizione degli attributi della struttura:

- **`Type ContactWarehousesType`**: rappresenta il tipo di operazione, che può essere un ordine[G] (**order**) o un trasferimento[G] (**transfer**);
- **`Order *ContactWarehousesOrder`**: rappresenta i dettagli dell'ordine da gestire, se l'operazione è di tipo ordine[G]
- **`Transfer *ContactWarehousesTransfer`**: rappresenta i dettagli del trasferimento[G] da gestire, se l'operazione è di tipo trasferimento[G]
- **`ConfirmedReservations []ConfirmedReservation`**: rappresenta una lista di prenotazioni confermate, utilizzate per ottimizzare la gestione delle risorse;
- **`ExcludeWarehouses []string`**: rappresenta una lista di identificativi dei magazzini da escludere dall'operazione;
- **`RetryUntil int64`**: rappresenta il *timestamp* fino al quale è possibile effettuare tentativi di contatto con i magazzini;
- **`RetryInTime int64`**: rappresenta l'intervallo di tempo, in millisecondi, tra un tentativo di contatto e il successivo.

3.4.8.1.31 - ContactWarehousesType

Rappresenta il tipo di operazione per il comando **ContactWarehousesCmd**.

Descrizione dei valori possibili:

- **`ContactWarehousesTypeOrder`**: indica che l'operazione è relativa a un ordine[G]
- **`ContactWarehousesTypeTransfer`**: indica che l'operazione è relativa a un trasferimento[G].

3.4.8.1.32 - ContactWarehousesOrder

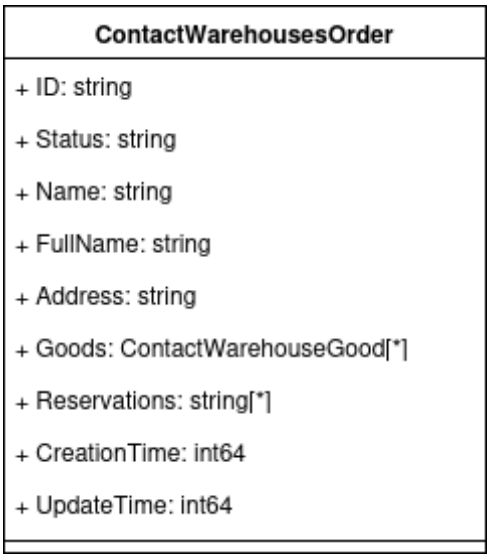

Figura 83: Order - ContactWarehousesOrder

Rappresenta i dettagli di un ordine[G] da gestire nel contesto del comando **ContactWarehousesCmd**.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine;
- **`Status string`**: rappresenta lo stato dell'ordine;
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`Goods []ContactWarehousesGood`**: rappresenta una lista di merci coinvolte nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine.

3.4.8.1.33 - ContactWarehousesTransfer

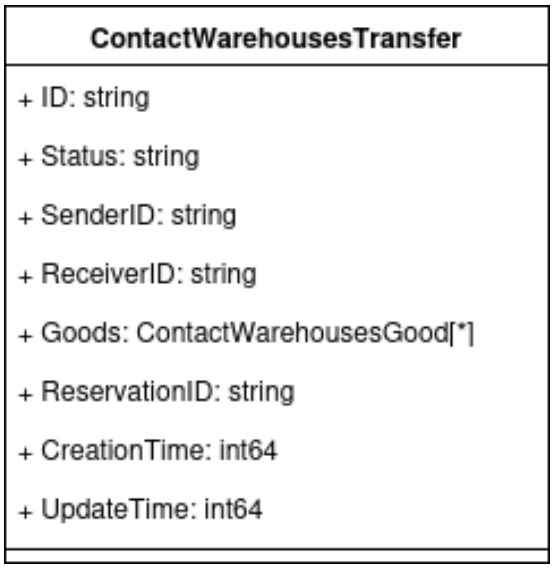

Figura 84: Order - ContactWarehousesTransfer

Rappresenta i dettagli di un trasferimento[G] da gestire nel contesto del comando **ContactWarehousesCmd**.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G]
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G]
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`Goods []ContactWarehousesGood`**: rappresenta una lista di merci coinvolte nel trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G].

3.4.8.1.34 - ContactWarehousesGood

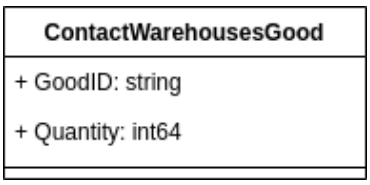

Figura 85: Order - ContactWarehousesGood

Rappresenta una merce coinvolta in un ordine[G] o trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce.

3.4.8.1.35 - ConfirmedReservation

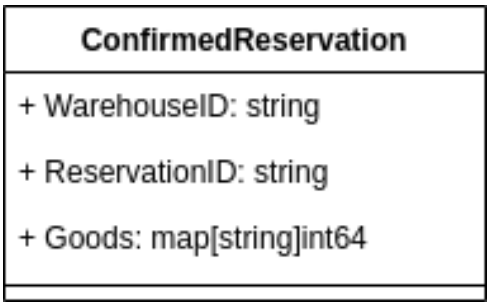

Figura 86: Order - ConfirmedReservation

Rappresenta una prenotazione confermata per un magazzino.

Descrizione degli attributi della struttura:

- **`WarehouseID string`**: rappresenta l'identificativo del magazzino associato alla prenotazione;
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione;
- **`Goods map[string]int64`**: rappresenta una mappa che associa l'identificativo della merce (**string**) alla quantità prenotata (**int64**).

3.4.8.1.36 - ContactWarehousesResponse

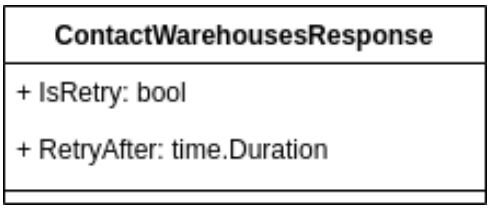

Figura 87: Order - ContactWarehousesResponse

Rappresenta la risposta alla richiesta di contatto con i magazzini.

Descrizione degli attributi della struttura:

- **`IsRetry bool`**: indica se è necessario effettuare un nuovo tentativo di contatto;
- **`RetryAfter time.Duration`**: rappresenta il tempo da attendere prima di effettuare un nuovo tentativo.

3.4.8.1.37 - SendOrderUpdateCmd

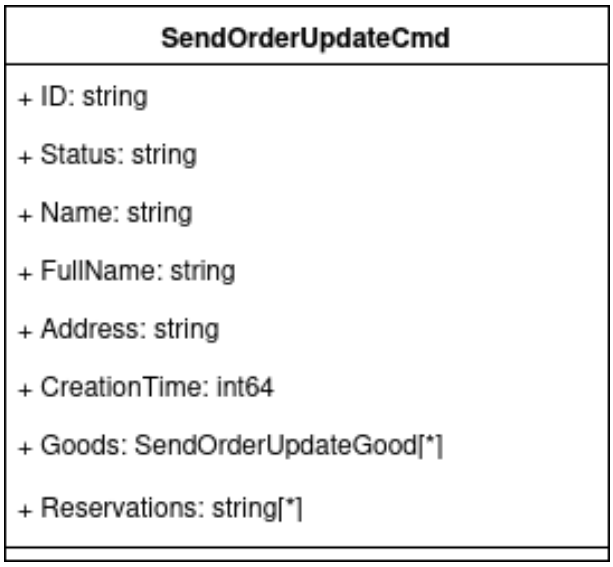

Figura 88: Order - SendOrderUpdateCmd

Rappresenta il comando utilizzato per inviare un aggiornamento di un ordine[G].

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'ordine da aggiornare;
- **`Status string`**: rappresenta lo stato dell'ordine (es. «Created», «Filled»);
- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione dell'ordine;
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento dell'ordine;
- **`Goods []SendOrderUpdateGood`**: rappresenta una lista di oggetti `SendOrderUpdateGood` che contengono le informazioni sulle merci incluse nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.38 - SendOrderUpdateGood

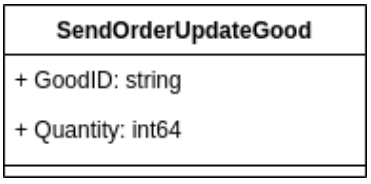

Figura 89: Order - SendOrderUpdateGood

Rappresenta una merce inclusa in un aggiornamento di un ordine[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce inclusa nell'ordine;
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.39 - SendContactWarehouseCmd

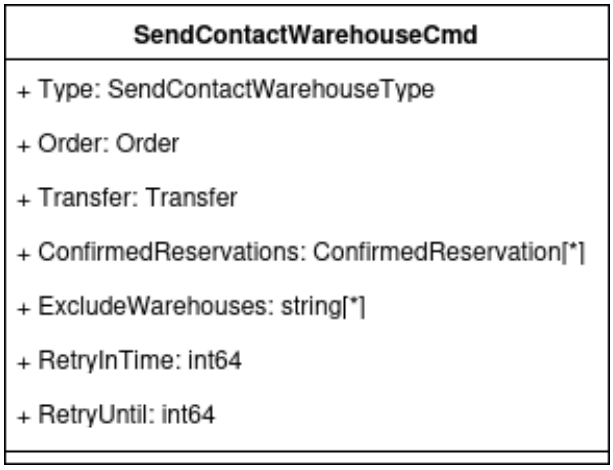

Figura 90: Order - SendContactWarehouseCmd

Rappresenta il comando utilizzato per contattare i magazzini al fine di gestire ordini o trasferimenti.

Descrizione degli attributi della struttura:

- **`Order *model.Order`**: rappresenta i dettagli dell'ordine da gestire, se l'operazione è di tipo ordine[G]
- **`Transfer *model.Transfer`**: rappresenta i dettagli del trasferimento[G] da gestire, se l'operazione è di tipo trasferimento[G]
- **`Type SendContactWarehouseType`**: rappresenta il tipo di operazione, che può essere un ordine[G] (**order**) o un trasferimento[G] (**transfer**);
- **`ConfirmedReservations []ConfirmedReservation`**: rappresenta una lista di prenotazioni confermate, utilizzate per ottimizzare la gestione delle risorse;
- **`ExcludeWarehouses []string`**: rappresenta una lista di identificativi dei magazzini da escludere dall'operazione;
- **`RetryInTime int64`**: rappresenta l'intervallo di tempo, in millisecondi, tra un tentativo di contatto e il successivo;
- **`RetryUntil int64`**: rappresenta il *timestamp* fino al quale è possibile effettuare tentativi di contatto con i magazzini.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.40 - SendContactWarehouseType

Rappresenta il tipo di operazione per il comando **SendContactWarehouseCmd**.

Descrizione dei valori possibili:

- **`SendContactWarehouseTypeOrder`**: indica che l'operazione è relativa a un ordine[G]
- **`SendContactWarehouseTypeTransfer`**: indica che l'operazione è relativa a un trasferimento[G].

3.4.8.1.41 - CreateOrderGood

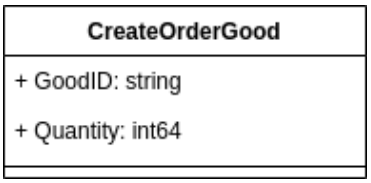

Figura 91: Order - CreateOrderGood

Rappresenta una merce inclusa in un ordine[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce inclusa nell'ordine;
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.42 - CreateOrderCmd

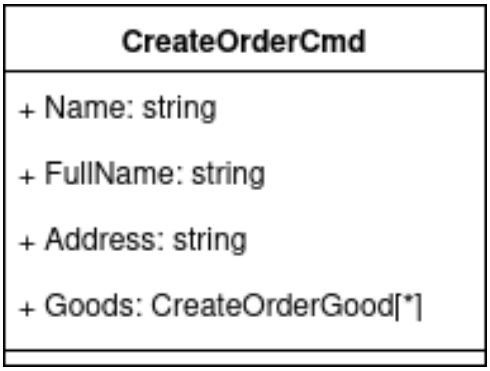

Figura 92: Order - CreateOrderCmd

Rappresenta il comando utilizzato per creare un nuovo ordine[G].

Descrizione degli attributi della struttura:

- **`Name string`**: rappresenta il nome dell'ordine;
- **`FullName string`**: rappresenta il nome completo del destinatario dell'ordine;
- **`Address string`**: rappresenta l'indirizzo del destinatario dell'ordine;
- **`Goods []CreateOrderGood`**: rappresenta una lista di oggetti `CreateOrderGood` che contengono le informazioni sulle merci incluse nell'ordine.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.43 - CreateOrderResponse

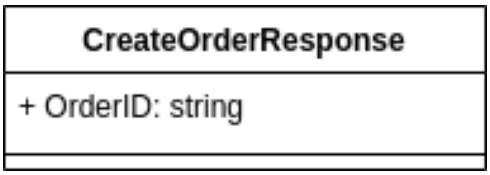

Figura 93: Order - CreateOrderResponse

Rappresenta la risposta alla richiesta di creazione di un nuovo ordine[G].

Descrizione degli attributi della struttura:

- **`OrderID string`**: rappresenta l'identificativo univoco dell'ordine creato.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.44 - GetOrderCmd

Rappresenta il comando utilizzato per richiedere i dettagli di un ordine[G] specifico nel sistema.

Descrizione degli attributi della struttura:

- **`string`**: rappresenta l'identificativo univoco dell'ordine richiesto.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.45 - GetTransferCmd

Rappresenta il comando utilizzato per richiedere i dettagli di un trasferimento[G] specifico nel sistema.

Descrizione degli attributi della struttura:

- **`string`**: rappresenta l'identificativo univoco del trasferimento[G] richiesto.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.46 - RequestedGood

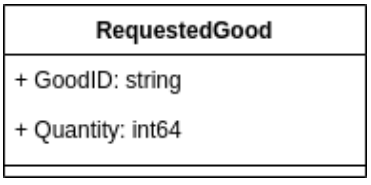

Figura 94: Order - RequestedGood

Rappresenta una merce richiesta per il calcolo della disponibilità.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce richiesta;
- **`Quantity int64`**: rappresenta la quantità richiesta della merce.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.47 - CalculateAvailabilityCmd

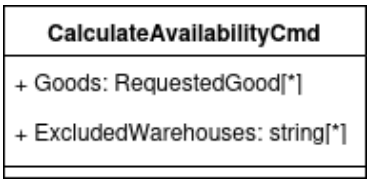

Figura 95: Order - CalculateAvailabilityCmd

Rappresenta il comando utilizzato per calcolare la disponibilità delle merci richieste nei magazzini.

Descrizione degli attributi della struttura:

- **`Goods []RequestedGood`**: rappresenta una lista di oggetti `RequestedGood` che contengono le informazioni sulle merci richieste e le rispettive quantità;
- **`ExcludedWarehouses []string`**: rappresenta una lista di identificativi dei magazzini da escludere dal calcolo della disponibilità.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.48 - WarehouseAvailability

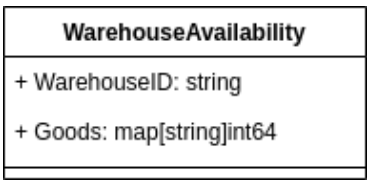

Figura 96: Order - WarehouseAvailability

Rappresenta la disponibilità delle merci in un magazzino specifico.

Descrizione degli attributi della struttura:

- **`WarehouseID string`**: rappresenta l'identificativo del magazzino;
- **`Goods map[string]int64`**: rappresenta una mappa che associa l'identificativo della merce (**string**) alla quantità disponibile (**int64**) nel magazzino.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.49 - CalculateAvailabilityResponse

Figura 97: Order - CalculateAvailabilityResponse

Rappresenta la risposta al comando di calcolo della disponibilità delle merci.

Descrizione degli attributi della struttura:

- **`Warehouses []WarehouseAvailability`**: rappresenta una lista di oggetti `WarehouseAvailability` che contengono le informazioni sulla disponibilità delle merci nei vari magazzini.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.50 - ApplyStockUpdateCmd

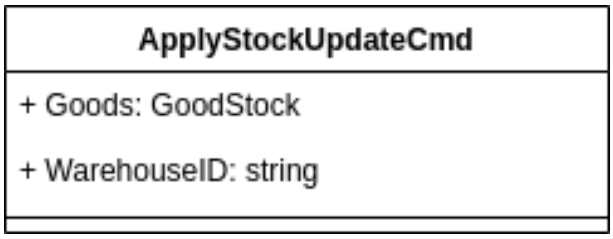

Figura 98: Order - ApplyStockUpdateCmd

Rappresenta il comando utilizzato per applicare un aggiornamento dello stock[G] in un magazzino specifico.

Descrizione degli attributi della struttura:

- **`WarehouseID string`**: rappresenta l'identificativo univoco del magazzino in cui applicare l'aggiornamento dello stock[G]
- **`Goods []model.GoodStock`**: rappresenta una lista di oggetti `GoodStock` che contengono le informazioni sulle merci e le rispettive quantità da aggiornare.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.51 - ReservationGood

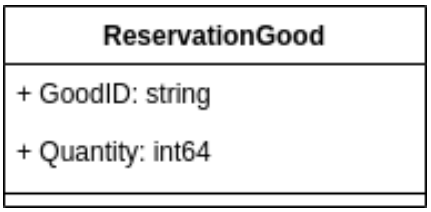

Figura 99: Order - ReservationGood

Rappresenta una merce coinvolta in una richiesta di prenotazione.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce da prenotare;
- **`Quantity int64`**: rappresenta la quantità della merce da prenotare.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.52 - RequestReservationCmd

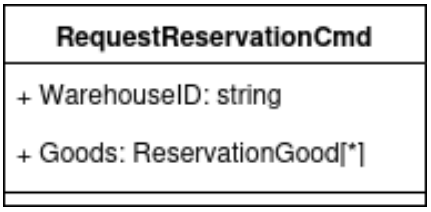

Figura 100: Order - RequestReservationCmd

Rappresenta il comando utilizzato per richiedere una prenotazione di merci in un magazzino.

Descrizione degli attributi della struttura:

- **`WarehouseID string`**: rappresenta l'identificativo del magazzino in cui effettuare la prenotazione;
- **`Goods []ReservationGood`**: rappresenta una lista di oggetti `ReservationGood` che contengono le informazioni sulle merci da prenotare.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.53 - RequestReservationResponse

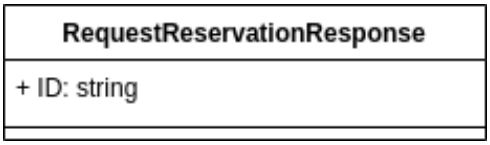

Figura 101: Order - RequestReservationResponse

Rappresenta la risposta alla richiesta di prenotazione di merci.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco della prenotazione creata.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.54 - SendTransferUpdateCmd

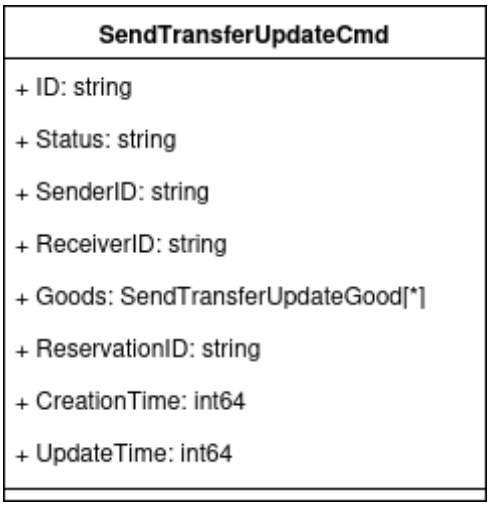

Figura 102: Order - SendTransferUpdateCmd

Rappresenta il comando utilizzato per inviare un aggiornamento di un trasferimento[G].

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del trasferimento[G] da aggiornare;
- **`Status string`**: rappresenta lo stato del trasferimento[G] (es. «Created», «Filled»);
- **`CreationTime int64`**: rappresenta il *timestamp* di creazione del trasferimento[G]
- **`UpdateTime int64`**: rappresenta il *timestamp* dell'ultimo aggiornamento del trasferimento[G]
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Goods []SendTransferUpdateGood`**: rappresenta una lista di oggetti `SendTransferUpdateGood` che contengono le informazioni sulle merci incluse nel trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.1.55 - SendTransferUpdateGood

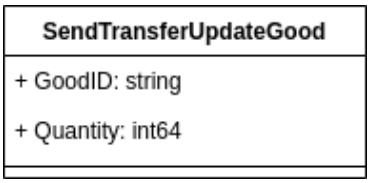

Figura 103: Order - SendTransferUpdateGood

Rappresenta una merce inclusa in un aggiornamento di un trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce inclusa nel trasferimento[G]
- **`Quantity int64`**: rappresenta la quantità della merce inclusa nel trasferimento[G].

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.8.2 - IGetStockPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere informazioni sulla quantità di una merce presente nel magazzino, la quantità globale di una merce o l'elenco dei magazzini.

Descrizione dei metodi dell'interfaccia:

- **`GetStock(GetStockCmd) (model.GoodStock, error)`**: il metodo deve permettere di ottenere la quantità totale di una merce presente in un magazzino specifico, prendendo come parametro un comando `GetStockCmd` e restituendo un oggetto `GoodStock` e un eventuale errore in caso di fallimento;
- **`GetGlobalStock(model.GoodID) model.GoodStock`**: il metodo deve permettere di ottenere la quantità globale di una merce presente in tutti i magazzini, prendendo come parametro l'identificativo della merce (`goodId`) e restituendo un oggetto `GoodStock`;
- **`GetWarehouses() []model.Warehouse`**: il metodo deve permettere di ottenere l'elenco di tutti i magazzini registrati nel sistema, restituendo una slice di oggetti `Warehouse`.

3.4.8.3 - SimpleCalculateAvailabilityService

La struttura `SimpleCalculateAvailabilityService` rappresenta un servizio per calcolare la disponibilità delle merci nei magazzini. Questo servizio utilizza un approccio semplice per determinare quali magazzini possono soddisfare le richieste di quantità delle merci, tenendo conto delle scorte disponibili.

Implementa le seguenti interfacce (*Use Case*):

- **ICalculateAvailabilityUseCase**, Sezione 3.4.8.7.

Descrizione degli attributi della struttura:

- **`getStockPort port.IGetStockPort`**: rappresenta la porta che consente alla *business logic* di comunicare con la *persistence logic* per ottenere informazioni sulle scorte di merci nei magazzini. Per maggiori dettagli, vedere la Sezione 3.4.8.2.

Descrizione dei metodi invocabili dalla struttura:

- **`NewSimpleCalculateAvailabilityService(getStockPort port.IGetStockPort) *SimpleCalculateAvailabilityService`**: costruttore della struttura. Inizializza l'attributo `getStockPort` con il valore fornito come parametro e restituisce un'istanza di `SimpleCalculateAvailabilityService`.
- **`GetAvailable(ctx context.Context, cmd port.CalculateAvailabilityCmd) (port.CalculateAvailabilityResponse, error)`**: metodo che calcola la disponibilità delle merci richieste nei magazzini. Prende come parametri il contesto e un comando `CalculateAvailabilityCmd` contenente le informazioni sulle merci richieste e i magazzini esclusi. Restituisce un oggetto `CalculateAvailabilityResponse` contenente i dettagli sulla disponibilità delle merci nei vari magazzini e un eventuale errore in caso di fallimento.

3.4.8.4 - ISendOrderUpdatePort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare con l'esterno per inviare un aggiornamento di un ordine[G].

Descrizione dei metodi dell'interfaccia:

- **`SendOrderUpdate(context.Context, SendOrderUpdateCmd) (model.Order, error)`**: il metodo deve permettere di inviare un aggiornamento di un ordine[G], prendendo come parametri il contesto e un comando `SendOrderUpdateCmd`. Deve restituire un oggetto `Order` contenente le informazioni aggiornate sull'ordine e un eventuale errore in caso di fallimento.

3.4.8.5 - ISendContactWarehousePort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare con l'esterno per inviare un comando di contatto con i magazzini.

Descrizione dei metodi dell'interfaccia:

- **`SendContactWarehouses(context.Context, SendContactWarehouseCmd) error`**: il metodo deve permettere di inviare un comando di contatto con i magazzini, prendendo come parametri il contesto e un oggetto `SendContactWarehouseCmd`. Deve restituire un errore in caso di fallimento.

3.4.8.6 - IRequestReservationPort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare con l'esterno per richiedere una prenotazione di merci.

Descrizione dei metodi dell'interfaccia:

- **`RequestReservation(context.Context, RequestReservationCmd) (RequestReservationResponse, error)`**: il metodo deve permettere di richiedere una prenotazione di merci, prendendo come parametri il contesto e un comando `RequestReservationCmd`. Deve restituire un oggetto `RequestReservationResponse` contenente l'identificativo della prenotazione creata e un eventuale errore in caso di fallimento.

3.4.8.7 - ICalculateAvailabilityUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di calcolare la disponibilità delle merci nei magazzini.

Descrizione dei metodi dell'interfaccia:

- **`GetAvailable(context.Context, CalculateAvailabilityCmd) (CalculateAvailabilityResponse, error)`**: il metodo deve permettere di calcolare la disponibilità delle merci richieste nei magazzini, prendendo come parametri il contesto e un comando `CalculateAvailabilityCmd`. Deve restituire un oggetto `CalculateAvailabilityResponse` contenente le informazioni sulla disponibilità delle merci e un eventuale errore in caso di fallimento.

3.4.8.8 - ManageOrderService

La struttura `ManageOrderService` rappresenta un servizio per la gestione degli ordini e dei trasferimenti nel sistema. Si occupa di orchestrare le operazioni necessarie per creare,

recuperare e gestire ordini e trasferimenti, interfacciandosi con la *business logic* e la *persistence logic* tramite le porte definite.

Implementa le seguenti interfacce (*Use Case*):

- **ICreateOrderUseCase**, Sezione 3.4.8.22;
- **ICreateTransferUseCase**, Sezione 3.4.8.24;
- **IGetOrderUseCase**, Sezione 3.4.8.21;
- **IGetTransferUseCase**, Sezione 3.4.8.25;
- **IContactWarehousesUseCase**, Sezione 3.4.8.29.

Descrizione degli attributi della struttura:

- **`getOrderPort port.IGetOrderPort`**: rappresenta la porta per ottenere informazioni sugli ordini. Per maggiori dettagli, vedere la Sezione 3.4.8.10;
- **`getTransferPort port.IGetTransferPort`**: rappresenta la porta per ottenere informazioni sui trasferimenti. Per maggiori dettagli, vedere la Sezione 3.4.8.11;
- **`sendOrderUpdatePort port.ISendOrderUpdatePort`**: rappresenta la porta per inviare aggiornamenti sugli ordini. Per maggiori dettagli, vedere la Sezione 3.4.8.4;
- **`sendTransferUpdatePort port.ISendTransferUpdatePort`**: rappresenta la porta per inviare aggiornamenti sui trasferimenti. Per maggiori dettagli, vedere la Sezione 3.4.8.19;
- **`sendContactWarehousePort port.ISendContactWarehousePort`**: rappresenta la porta per contattare i magazzini. Per maggiori dettagli, vedere la Sezione 3.4.8.5;
- **`requestReservationPort port.IRequestReservationPort`**: rappresenta la porta per richiedere prenotazioni di merci. Per maggiori dettagli, vedere la Sezione 3.4.8.6;
- **`calculateAvailabilityUseCase port.ICalculateAvailabilityUseCase`**: rappresenta il caso d'uso[G] per calcolare la disponibilità delle merci nei magazzini. Per maggiori dettagli, vedere la Sezione 3.4.8.7;
- **`transactionPort port.ITransactionPort`**: rappresenta la porta per gestire operazioni transazionali. Per maggiori dettagli, vedere la Sezione 3.4.8.14.

Descrizione dei metodi invocabili dalla struttura:

- **`NewManageOrderService(p ManageOrderServiceParams) *ManageOrderService`**: costruttore della struttura. Inizializza gli attributi con i valori forniti tramite `ManageOrderServiceParams` e restituisce un'istanza di `ManageOrderService`;
- **`CreateOrder(ctx context.Context, cmd port.CreateOrderCmd) (port.CreateOrderResponse, error)`**: metodo per creare un nuovo ordine[G]. Genera un identificativo univoco per l'ordine, salva i dettagli dell'ordine e avvia il processo di contatto con i magazzini;
- **`GetOrder(ctx context.Context, orderId string) (model.Order, error)`**: metodo per ottenere i dettagli di un ordine[G] specifico tramite il suo identificativo;
- **`GetAllOrders(ctx context.Context) []model.Order`**: metodo per ottenere una lista di tutti gli ordini registrati nel sistema;
- **`CreateTransfer(ctx context.Context, cmd port.CreateTransferCmd) (port.CreateTransferResponse, error)`**: metodo per creare un nuovo trasferimento[G]. Genera un identificativo univoco per il trasferimento[G], salva i dettagli del trasferimento[G] e avvia il processo di contatto con i magazzini;

- **`GetTransfer(ctx context.Context, transferId string) (model.Transfer, error)`**: metodo per ottenere i dettagli di un trasferimento[G] specifico tramite il suo identificativo;
- **`GetAllTransfers(ctx context.Context) []model.Transfer`**: metodo per ottenere una lista di tutti i trasferimenti registrati nel sistema;
- **`ContactWarehouses(ctx Context, cmd port.ContactWarehousesCmd) (port.ContactWarehousesResponse, error)`**: metodo per contattare i magazzini al fine di gestire ordini o trasferimenti. Gestisce i tentativi di contatto e le prenotazioni delle merci necessarie;
- **`createOrderCmdToSendOrderUpdateCmd(orderId string, cmd port.CreateOrderCmd) port.SendOrderUpdateCmd`**: funzione di utilità per convertire un comando di creazione ordine[G] in un comando di aggiornamento ordine[G]
- **`contactCmdToCalculateAvailabilityCmd(cmd port.ContactWarehousesCmd) port.CalculateAvailabilityCmd`**: funzione di utilità per convertire un comando di contatto con i magazzini in un comando di calcolo della disponibilità;
- **`warehouseAvailabilityToReservationCmd(warehouse port.WarehouseAvailability) port.RequestReservationCmd`**: funzione di utilità per convertire la disponibilità di un magazzino in un comando di prenotazione;
- **`contactCmdAndConfirmedToSendOrderUpdateCmd(cmd port.ContactWarehousesCmd, confirmed []port.ConfirmedReservation) port.SendOrderUpdateCmd`**: funzione di utilità per convertire un comando di contatto con i magazzini e le prenotazioni confermate in un comando di aggiornamento ordine[G]
- **`contactCmdToSendOrderUpdateCmdForCancel(cmd port.ContactWarehousesCmd) port.SendOrderUpdateCmd`**: funzione di utilità per convertire un comando di contatto con i magazzini in un comando di aggiornamento ordine[G] per annullare l'ordine.

3.4.8.9 - IApplyStockUpdatePort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento dello stock[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyStockUpdate(ApplyStockUpdateCmd) error`**: il metodo deve permettere di applicare un aggiornamento dello stock[G]. Prende come parametro un oggetto `ApplyStockUpdateCmd` che contiene tutte le informazioni necessarie per l'aggiornamento dello stock[G]. Deve restituire un errore in caso di fallimento.

3.4.8.10 - IGetOrderPort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere informazioni sugli ordini.

Descrizione dei metodi dell'interfaccia:

- **`GetOrder(model.OrderID) (model.Order, error)`**: il metodo deve permettere di ottenere i dettagli di un ordine[G] specifico, prendendo come parametro l'identificativo dell'ordine (`orderId`). Deve restituire un oggetto `Order` e un eventuale errore in caso di fallimento;
- **`GetAllOrder() []Order`**: il metodo deve permettere di ottenere una lista di tutti gli ordini registrati nel sistema, restituendo una slice di oggetti `Order`.

3.4.8.11 - IGetTransferPort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere informazioni sui trasferimenti.

Descrizione dei metodi dell'interfaccia:

- **`GetTransfer(model.TransferID) (model.Transfer, error)`**: il metodo deve permettere di ottenere i dettagli di un trasferimento[G] specifico, prendendo come parametro l'identificativo del trasferimento[G] (`transferId`). Deve restituire un oggetto `Transfer` e un eventuale errore in caso di fallimento;
- **`GetAllTransfer() []model.Transfer`**: il metodo deve permettere di ottenere una lista di tutti i trasferimenti registrati nel sistema, restituendo una slice di oggetti `Transfer`.

3.4.8.12 - ISetCompleteTransferPort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di completare un trasferimento[G] o incrementare il numero di aggiornamenti dello stock[G] associati a un trasferimento[G].

Descrizione dei metodi dell'interfaccia:

- **`SetComplete(model.TransferID) error`**: il metodo deve permettere di segnare un trasferimento[G] come completato, prendendo come parametro l'identificativo del trasferimento[G] (`transferId`). Deve restituire un errore in caso di fallimento;
- **`IncrementLinkedStockUpdate(model.TransferID) error`**: il metodo deve permettere di incrementare il numero di aggiornamenti dello stock[G] associati a un trasferimento[G], prendendo come parametro l'identificativo del trasferimento[G] (`transferId`). Deve restituire un errore in caso di fallimento.

3.4.8.13 - ISetCompletedWarehouseOrderPort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di completare un ordine[G] o segnalare il completamento di un ordine[G] da parte di un magazzino.

Descrizione dei metodi dell'interfaccia:

- **`SetCompletedWarehouse(SetCompletedWarehouseCmd) (model.Order, error)`**: il metodo deve permettere di segnalare il completamento di un ordine[G] da parte di un magazzino, prendendo come parametro un comando `SetCompletedWarehouseCmd`. Deve restituire un oggetto `Order` e un eventuale errore in caso di fallimento;
- **`SetComplete(model.OrderID) error`**: il metodo deve permettere di segnare un ordine[G] come completato, prendendo come parametro l'identificativo dell'ordine (`orderId`). Deve restituire un errore in caso di fallimento.

3.4.8.14 - ITransactionPort

Rappresenta l'interfaccia che consente alla *business logic* di gestire operazioni transazionali, fornendo metodi per bloccare e sbloccare risorse condivise.

Descrizione dei metodi dell'interfaccia:

- **`Lock()`**: il metodo deve permettere di bloccare una risorsa condivisa, garantendo che solo una transazione possa accedervi alla volta.

- **`Unlock()`**: il metodo deve permettere di sbloccare una risorsa condivisa, consentendo ad altre transazioni di accedervi.

3.4.8.15 - TransactionImpl

La struttura `TransactionImpl` rappresenta un'implementazione dell'interfaccia `ITransactionPort` per la gestione delle operazioni transazionali nel microservizio **Order**[G]. Utilizza un mutex per garantire l'accesso esclusivo alle risorse condivise.

Descrizione degli attributi della struttura:

- **`m sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente.

Descrizione dei metodi invocabili dalla struttura:

- **`NewTransactionImpl() *TransactionImpl`**: costruttore della struttura. Inizializza l'attributo `m` con un nuovo mutex e restituisce un'istanza di `TransactionImpl`.
- **`Lock()`**: blocca il mutex, garantendo che solo una transazione possa accedere alla risorsa condivisa alla volta.
- **`Unlock()`**: sblocca il mutex, consentendo ad altre transazioni di accedere alla risorsa condivisa.

3.4.8.16 - ApplyStockUpdateService

La struttura `ApplyStockUpdateService` rappresenta un servizio per gestire l'applicazione degli aggiornamenti relativi allo stock[G]. Questo servizio si occupa di elaborare gli aggiornamenti dello stock[G] provenienti da ordini o trasferimenti, applicandoli al magazzino e aggiornando lo stato degli ordini o dei trasferimenti associati.

Implementa le seguenti interfacce (*Use Case*):

- **IApplyStockUpdateUseCase**, Sezione 3.4.8.31.

Descrizione degli attributi della struttura:

- **`applyStockUpdatePort port.IApplyStockUpdatePort`**: rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento dello stock[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.9;
- **`applyOrderUpdatePort port.IApplyOrderUpdatePort`**: rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento di un ordine[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.17;
- **`getOrderPort port.IGetOrderPort`**: rappresenta la porta che consente alla *business logic* di ottenere informazioni sugli ordini. Per maggiori informazioni, vedere la Sezione 3.4.8.10;
- **`getTransferPort port.IGetTransferPort`**: rappresenta la porta che consente alla *business logic* di ottenere informazioni sui trasferimenti. Per maggiori informazioni, vedere la Sezione 3.4.8.11;
- **`applyTransferUpdatePort port.IApplyTransferUpdatePort`**: rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento di un trasferimento[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.18;
- **`setCompleteTransferPort port.ISetCompleteTransferPort`**: rappresenta la porta che consente alla *business logic* di segnare un trasferimento[G] come completato o

incrementare il numero di aggiornamenti dello stock[G] associati a un trasferimento[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.12;
- **`setCompletedWarehousePort port.ISetCompletedWarehouseOrderPort`**: rappresenta la porta che consente alla *business logic* di segnalare il completamento di un ordine[G] da parte di un magazzino o di segnare un ordine[G] come completato. Per maggiori informazioni, vedere la Sezione 3.4.8.13;
- **`transactionPort port.ITransactionPort`**: rappresenta la porta che consente alla *business logic* di gestire operazioni transazionali, fornendo metodi per bloccare e sbloccare risorse condivise. Per maggiori informazioni, vedere la Sezione 3.4.8.14.

Descrizione dei metodi invocabili dalla struttura:

- **`NewApplyStockUpdateService(p ApplyStockUpdateServiceParams) *ApplyStockUpdateService`**: costruttore della struttura. Inizializza gli attributi con i valori forniti tramite `ApplyStockUpdateServiceParams` e restituisce un'istanza di `ApplyStockUpdateService`;
- **`ApplyStockUpdate(ctx context.Context, cmd port.StockUpdateCmd) error`**: metodo principale che gestisce l'applicazione di un aggiornamento dello stock[G]. Controlla se l'aggiornamento è relativo a un ordine[G] o a un trasferimento[G] e chiama i metodi appropriati per elaborarlo. Successivamente, applica l'aggiornamento dello stock[G] al magazzino tramite la porta `applyStockUpdatePort`;
- **`applyStockUpdateFromTransfer(cmd port.StockUpdateCmd) error`**: metodo che gestisce l'applicazione di un aggiornamento dello stock[G] relativo a un trasferimento[G]. Aggiorna lo stato del trasferimento[G] e verifica[G] se il trasferimento[G] è completato;
- **`applyStockUpdateFromOrder(cmd port.StockUpdateCmd) error`**: metodo che gestisce l'applicazione di un aggiornamento dello stock[G] relativo a un ordine[G]. Aggiorna lo stato dell'ordine e verifica[G] se l'ordine è completato.

3.4.8.17 - IApplyOrderUpdatePort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento di un ordine[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyOrderUpdate(ApplyOrderUpdateCmd) error`**: il metodo deve permettere di applicare un aggiornamento di un ordine[G], prendendo come parametro un oggetto di tipo `ApplyOrderUpdateCmd`. Deve restituire un errore in caso di fallimento.

3.4.8.18 - IApplyTransferUpdatePort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento di un trasferimento[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyTransferUpdate(ApplyTransferUpdateCmd) error`**: il metodo deve permettere di applicare un aggiornamento di un trasferimento[G], prendendo come parametro un oggetto di tipo `ApplyTransferUpdateCmd`. Deve restituire un errore in caso di fallimento.

3.4.8.19 - ISendTransferUpdatePort

Rappresenta l'interfaccia che permette alla *business logic* di comunicare con l'esterno per inviare un aggiornamento di un trasferimento[G].

Descrizione dei metodi dell'interfaccia:

- **`SendTransferUpdate(context.Context, SendTransferUpdateCmd) (model.Transfer, error)`**: il metodo deve permettere di inviare un aggiornamento di un trasferimento[G], prendendo come parametri il contesto e un comando `SendTransferUpdateCmd`. Deve restituire un oggetto `Transfer` contenente le informazioni aggiornate sul trasferimento[G] e un eventuale errore in caso di fallimento.

3.4.8.20 - ApplyOrderUpdateService

La struttura `ApplyOrderUpdateService` rappresenta un servizio per gestire l'applicazione degli aggiornamenti relativi agli ordini e ai trasferimenti. Questo servizio si occupa di convertire i comandi ricevuti in un formato compatibile con le porte della *persistence logic* e di inoltrare tali comandi per l'elaborazione.

Implementa le seguenti interfacce (*Use Case*):

- **IApplyOrderUpdateUseCase**, Sezione 3.4.8.27;
- **IApplyTransferUpdateUseCase**, Sezione 3.4.8.28.

Descrizione degli attributi della struttura:

- **`applyOrderUpdatePort port.IApplyOrderUpdatePort`**: rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento di un ordine[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.17;
- **`applyTransferUpdatePort port.IApplyTransferUpdatePort`**: rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento di un trasferimento[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.18;
- **`transactionPort port.ITransactionPort`**: rappresenta la porta che consente alla *business logic* di gestire operazioni transazionali, fornendo metodi per bloccare e sbloccare risorse condivise. Per maggiori informazioni, vedere la Sezione 3.4.8.14.

Descrizione dei metodi invocabili dalla struttura:

- **`NewApplyOrderUpdateService(p ApplyOrderUpdateServiceParams) *ApplyOrderUpdateService`**: costruttore della struttura. Inizializza gli attributi `applyOrderUpdatePort` e `applyTransferUpdatePort` con i valori forniti mediante `ApplyOrderUpdateServiceParams`, struttura dotata di `fx.In` e restituisce un'istanza di `ApplyOrderUpdateService`;
- **`ApplyOrderUpdate(ctx context.Context, cmd port.OrderUpdateCmd)`**: metodo che gestisce l'applicazione di un aggiornamento di un ordine[G]. Converte il comando `OrderUpdateCmd` in un comando `ApplyOrderUpdateCmd` utilizzando la funzione `orderUpdateCmdToApplyOrderUpdateCmd` e lo inoltra alla porta `applyOrderUpdatePort` per l'elaborazione;
- **`ApplyTransferUpdate(ctx context.Context, cmd port.TransferUpdateCmd)`**: metodo che gestisce l'applicazione di un aggiornamento di un trasferimento[G]. Converte il comando `TransferUpdateCmd` in un comando `ApplyTransferUpdateCmd` utilizzando la funzione `transferUpdateCmdToApplyTransferUpdateCmd` e lo inoltra alla porta `applyTransferUpdatePort` per l'elaborazione;
- **`orderUpdateCmdToApplyOrderUpdateCmd(cmd port.OrderUpdateCmd) port.ApplyOrderUpdateCmd`**: funzione di utilità che converte un comando `OrderUpdateCmd`

in un comando `ApplyOrderUpdateCmd`. Mappa le informazioni sulle merci e i relativi dettagli dal comando originale al nuovo comando, rendendolo pronto per essere elaborato dalla *persistence logic*;

- **`transferUpdateCmdToApplyTransferUpdateCmd(cmd port.TransferUpdateCmd) port.ApplyTransferUpdateCmd`**: funzione di utilità che converte un comando `TransferUpdateCmd` in un comando `ApplyTransferUpdateCmd`. Mappa le informazioni sulle merci e i relativi dettagli dal comando originale al nuovo comando, rendendolo pronto per essere elaborato dalla *persistence logic*.

3.4.8.21 - IGetOrderUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di ottenere informazioni sugli ordini.

Descrizione dei metodi dell'interfaccia:

- **`GetOrder(ctx context.Context, GetOrderCmd) (model.Order, error)`**: il metodo deve permettere di ottenere i dettagli di un ordine[G] specifico, prendendo come parametri il contesto e un comando `GetOrderCmd`. Deve restituire un oggetto `Order` e un eventuale errore in caso di fallimento;
- **`GetAllOrders(ctx context.Context) []model.Order`**: il metodo deve permettere di ottenere una lista di tutti gli ordini registrati nel sistema, prendendo come parametro il contesto e restituendo una slice di oggetti `Order`.

3.4.8.22 - ICreateOrderUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di creare un nuovo ordine[G].

Descrizione dei metodi dell'interfaccia:

- **`CreateOrder(ctx context.Context, cmd CreateOrderCmd) (CreateOrderResponse, error)`**: il metodo deve permettere di creare un nuovo ordine[G], prendendo come parametri il contesto e un comando `CreateOrderCmd`. Deve restituire un oggetto `CreateOrderResponse` contenente l'identificativo dell'ordine creato e un eventuale errore in caso di fallimento.

3.4.8.23 - OrderController

La struttura `OrderController` rappresenta l'*application logic* del microservizio **Order**. Si occupa di gestire le richieste relative alla creazione, al recupero e alla gestione degli ordini, interfacciandosi con la *business logic* attraverso i relativi *Use Case*.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`**: rappresenta il broker di messaggistica NATS[G] utilizzato per pubblicare e ricevere messaggi.
- **`createOrderUseCase port.ICreateOrderUseCase`**: interfaccia per la creazione di un ordine[G].
- **`getOrderUseCase port.IGetOrderUseCase`**: interfaccia per il recupero degli ordini.

Descrizione dei metodi invocabili dalla struttura:

- **`NewOrderController(p OrderControllerParams) *OrderController`**: costruttore della struttura. Inizializza gli attributi `createOrderUseCase` e `getOrderUseCase` con i valori forniti tramite `OrderControllerParams`, che contiene anche quanto necessario per la telemetria.

- **`OrderCreateHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di creazione di un ordine[G]. Valida i dati della richiesta, invoca il *Use Case* per la creazione dell'ordine e risponde con l'esito dell'operazione.
- **`OrderGetHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di recupero di un ordine[G] specifico. Valida i dati della richiesta, invoca il *Use Case* per il recupero dell'ordine e risponde con i dettagli dell'ordine.
- **`OrderGetAllHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di recupero di tutti gli ordini. Invoca il *Use Case* per ottenere la lista degli ordini e risponde con i dettagli.
- **`checkCreateOrderRequestDTO(dto request.CreateOrderRequestDTO) error`**: valida i dati della richiesta di creazione di un ordine[G]. Restituisce un errore se i dati non sono validi.
- **`orderToGetGoodResponseDTO(order Order) response.GetOrderResponseDTO`**: converte un oggetto `Order` in un oggetto `response.GetOrderResponseDTO`.
- **`ordersToGetAllGoodResponseDTO(model []Order) response.GetAllOrderResponseDTO`**: converte una lista di oggetti `Order` in un oggetto `response.GetAllOrderResponseDTO`.

3.4.8.24 - ICreateTransferUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di creare un nuovo trasferimento[G].

Descrizione dei metodi dell'interfaccia:

- **`CreateTransfer(context.Context, CreateTransferCmd) (CreateTransferResponse, error)`**: il metodo deve permettere di creare un nuovo trasferimento[G], prendendo come parametri il contesto e un comando `CreateTransferCmd`. Deve restituire un oggetto `CreateTransferResponse` contenente l'identificativo del trasferimento[G] creato e un eventuale errore in caso di fallimento.

3.4.8.25 - IGetTransferUseCase

Interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di ottenere informazioni sui trasferimenti.

Descrizione dei metodi dell'interfaccia:

- **`GetTransfer(context.Context, GetTransferCmd) (model.Transfer, error)`**: il metodo deve permettere di ottenere i dettagli di un trasferimento[G] specifico, prendendo come parametri il contesto e un comando `GetTransferCmd`. Deve restituire un oggetto `Transfer` e un eventuale errore in caso di fallimento;
- **`GetAllTransfers(context.Context) []model.Transfer`**: il metodo deve permettere di ottenere una lista di tutti i trasferimenti registrati nel sistema, prendendo come parametro il contesto e restituendo una slice di oggetti `Transfer`.

3.4.8.26 - TransferController

La struttura `TransferController` rappresenta l'*application logic* del microservizio **Order**. Si occupa di gestire le richieste relative alla creazione, al recupero e alla gestione dei trasferimenti, interfacciandosi con la *business logic* attraverso i relativi *Use Case*.

Descrizione degli attributi della struttura:

- **`createTransferUseCase port.ICreateTransferUseCase`**: interfaccia per la creazione di un trasferimento[G].
- **`getTransferUseCase port.IGetTransferUseCase`**: interfaccia per il recupero dei trasferimenti.

Descrizione dei metodi invocabili dalla struttura:

- **`NewTransferController(p TransferControllerParams) *TransferController`**: costruttore della struttura. Inizializza gli attributi `createTransferUseCase` e `getTransferUseCase` con i valori forniti tramite `TransferControllerParams`, che contiene anche quanto necessario per la telemetria.
- **`TransferCreateHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di creazione di un trasferimento[G]. Valida i dati della richiesta, invoca il *Use Case* per la creazione del trasferimento[G] e risponde con l'esito dell'operazione.
- **`TransferGetHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di recupero di un trasferimento[G] specifico. Valida i dati della richiesta, invoca il *Use Case* per il recupero del trasferimento[G] e risponde con i dettagli del trasferimento[G].
- **`TransferGetAllHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di recupero di tutti i trasferimenti. Invoca il *Use Case* per ottenere la lista dei trasferimenti e risponde con i dettagli.
- **`modelTransferToTransferInfoDTO(transfer Transfer) response.TransferInfo`**: converte un oggetto `Transfer` in un oggetto `response.TransferInfo`.

3.4.8.27 - IApplyOrderUpdateUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di applicare un aggiornamento di un ordine[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyOrderUpdate(context.Context, OrderUpdateCmd) error`**: il metodo deve permettere di applicare un aggiornamento di un ordine[G], prendendo come parametri il contesto e un oggetto di tipo `OrderUpdateCmd`. Deve restituire un errore in caso di fallimento.

3.4.8.28 - IApplyTransferUpdateUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di applicare un aggiornamento di un trasferimento[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyTransferUpdate(context.Context, TransferUpdateCmd)`**: il metodo deve permettere di applicare un aggiornamento di un trasferimento[G], prendendo come parametri il contesto e un oggetto di tipo `TransferUpdateCmd`. Deve restituire un errore in caso di fallimento.

3.4.8.29 - IContactWarehousesUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di contattare i magazzini per gestire ordini o trasferimenti.

Descrizione dei metodi dell'interfaccia:

- **`ContactWarehouses(context.Context, ContactWarehousesCmd) (ContactWarehousesResponse, error)`**: il metodo deve permettere di contattare i magazzini, prendendo come parametri il contesto e un comando `ContactWarehousesCmd`. Deve restituire un oggetto `ContactWarehousesResponse` contenente le informazioni sull'esito dell'operazione e un eventuale errore in caso di fallimento.

3.4.8.30 - OrderListener

La struttura `OrderListener` rappresenta un componente dell'*application logic* del microservizio **Order**. È progettata per ascoltare eventi relativi agli aggiornamenti di ordini, trasferimenti e contatti con i magazzini, provenienti dal sistema di messaggistica NATS[G] JetStream, e per inoltrare tali eventi alla *business logic* per l'elaborazione.

Descrizione degli attributi della struttura:

- **`applyOrderUpdateUseCase port.IApplyOrderUpdateUseCase`**: rappresenta il caso d'uso[G] per applicare aggiornamenti agli ordini. Per maggiori informazioni, vedere la Sezione 3.4.8.27;
- **`applyTransferUpdateUseCase port.IApplyTransferUpdateUseCase`**: rappresenta il caso d'uso[G] per applicare aggiornamenti ai trasferimenti. Per maggiori informazioni, vedere la Sezione 3.4.8.28;
- **`contactWarehouseUseCase port.IContactWarehousesUseCase`**: rappresenta il caso d'uso[G] per contattare i magazzini. Per maggiori informazioni, vedere la Sezione 3.4.8.29.

Descrizione dei metodi invocabili dalla struttura:

- **`NewOrderListener(p OrderListenerParams) *OrderListener`**: costruttore della struttura. Inizializza gli attributi con i valori forniti nella struttura `OrderListenerParams`, che contiene anche quanto necessario per la telemetria, e restituisce un'istanza di `OrderListener`;
- **`ListenOrderUpdate(ctx context.Context, msg jetstream.Msg) error`**: metodo che gestisce gli eventi di aggiornamento degli ordini. Riceve un messaggio dal sistema di messaggistica NATS[G] JetStream, lo deserializza in un oggetto `stream.OrderUpdate`, lo converte in un comando `OrderUpdateCmd` e lo inoltra alla *business logic* tramite il metodo `ApplyOrderUpdate` dell'interfaccia `IApplyOrderUpdateUseCase`. Restituisce un errore in caso di fallimento durante una qualsiasi di queste operazioni;
- **`ListenTransferUpdate(ctx context.Context, msg jetstream.Msg) error`**: metodo che gestisce gli eventi di aggiornamento dei trasferimenti. Riceve un messaggio dal sistema di messaggistica NATS[G] JetStream, lo deserializza in un oggetto `stream.TransferUpdate`, lo converte in un comando `TransferUpdateCmd` e lo inoltra alla *business logic* tramite il metodo `ApplyTransferUpdate` dell'interfaccia `IApplyTransferUpdateUseCase`. Restituisce un errore in caso di fallimento durante una qualsiasi di queste operazioni;
- **`ListenContactWarehouses(ctx Context, msg jetstream.Msg) error`**: metodo che gestisce gli eventi di contatto con i magazzini. Riceve un messaggio dal sistema di messaggistica NATS[G] JetStream, lo deserializza in un oggetto `internalStream.ContactWarehouses`, lo converte in un comando `ContactWarehousesCmd` e lo inoltra alla *business logic* tramite il metodo `ContactWarehouses` dell'interfaccia `IContactWarehousesUseCase`. Se è necessario un nuovo tentativo, utilizza il metodo `NakWithDelay` per ritardare il messaggio e restituisce un errore specifico per indicare che il messaggio non è stato riconosciuto;

- **`orderUpdateEventToApplyOrderUpdateCmd(event stream.OrderUpdate) port.OrderUpdateCmd`**: funzione di utilità che converte un evento `stream.OrderUpdate` in un comando `OrderUpdateCmd`. Mappa le informazioni sulle merci e i relativi dettagli dall'evento al comando, rendendolo pronto per essere elaborato dalla *business logic*;
- **`transferUpdateEventToApplyTransferUpdateCmd(event stream.TransferUpdate) port.TransferUpdateCmd`**: funzione di utilità che converte un evento `stream.TransferUpdate` in un comando `TransferUpdateCmd`. Mappa le informazioni sulle merci e i relativi dettagli dall'evento al comando, rendendolo pronto per essere elaborato dalla *business logic*.

3.4.8.31 - IApplyStockUpdateUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di applicare un aggiornamento dello *stock*[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyStockUpdate(context.Context,StockUpdateCmd) error`**: il metodo deve permettere di applicare un aggiornamento dello stock[G]. Prende come parametri il contesto e un oggetto `StockUpdateCmd` che contiene tutte le informazioni necessarie per l'aggiornamento dello stock[G].

3.4.8.32 - StockUpdateListener

La struttura `StockUpdateListener` rappresenta un componente dell'*application logic* del microservizio **Warehouse**[G]. È progettata per ascoltare eventi di aggiornamento dello stock[G] provenienti dal sistema di messaggistica NATS[G] JetStream e per inoltrare tali eventi alla *business logic* per l'elaborazione.

Descrizione degli attributi della struttura:

- **`applyStockUpdateUseCase port.IApplyStockUpdateUseCase`**: rappresenta l'interfaccia che consente alla *application logic* di comunicare con la *business logic* per applicare un aggiornamento dello stock[G]. Per maggiori informazioni, vedere la Sezione 3.4.8.31.

Descrizione dei metodi invocabili dalla struttura:

- **`NewStockUpdateListener(applyStockUpdateUseCase port.IApplyStockUpdateUseCase, mp MetricParams) *StockUpdateListener`**: costruttore della struttura. Inizializza l'attributo `applyStockUpdateUseCase` con il valore fornito come parametro e la telemetria con quanto contenuto in `MetricParams` e restituisce un'istanza di `StockUpdateListener`.
- **`ListenStockUpdate(ctx context.Context, msg jetstream.Msg) error`**: metodo che gestisce gli eventi di aggiornamento dello stock[G]. Riceve un messaggio dal sistema di messaggistica NATS[G] JetStream, lo deserializza in un oggetto `stream.StockUpdate`, lo converte in un comando `StockUpdateCmd` e lo inoltra alla *business logic* tramite il metodo `ApplyStockUpdate` dell'interfaccia `IApplyStockUpdateUseCase`. Restituisce un errore in caso di fallimento durante una qualsiasi di queste operazioni.
- **`StockUpdateEventToApplyStockUpdateCmd(event stream.StockUpdate) port.StockUpdateCmd`**: funzione di utilità che converte un evento `stream.StockUpdate` in un comando `StockUpdateCmd`. Mappa le informazioni sulle merci e i relativi dettagli dall'evento al comando, rendendolo pronto per essere elaborato dalla *business logic*.

3.4.8.33 - ITransferRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* per la gestione dei trasferimenti nel microservizio **Order**.

Descrizione dei metodi dell'interfaccia:

- **`GetTransfer(transferId string) (Transfer, error)`**: il metodo deve permettere di ottenere i dettagli di un trasferimento[G] specifico, prendendo come parametro l'identificativo del trasferimento[G] (`transferId`). Deve restituire un oggetto `Transfer` contenente i dettagli del trasferimento[G] e un eventuale errore in caso di fallimento;
- **`GetTransfers() []Transfer`**: il metodo deve permettere di ottenere una lista di tutti i trasferimenti registrati nel sistema. Deve restituire una slice di oggetti `Transfer`;
- **`SetTransfer(transferId string, transfer Transfer) bool`**: il metodo deve permettere di impostare o aggiornare i dettagli di un trasferimento[G] specifico, prendendo come parametri l'identificativo del trasferimento[G] (`transferId`) e un oggetto `Transfer` contenente i dettagli aggiornati. Deve restituire `true` se l'operazione è andata a buon fine, `false` altrimenti;
- **`SetComplete(transferId string) error`**: il metodo deve permettere di segnare un trasferimento[G] come completato, prendendo come parametro l'identificativo del trasferimento[G] (`transferId`). Deve restituire un errore in caso di fallimento;
- **`IncrementLinkedStockUpdate(transferId string) error`**: il metodo deve permettere di incrementare il numero di aggiornamenti dello stock[G] associati a un trasferimento[G], prendendo come parametro l'identificativo del trasferimento[G] (`transferId`). Deve restituire un errore in caso di fallimento.

3.4.8.34 - TransferRepositoryImpl

La struttura `TransferRepositoryImpl` rappresenta l'implementazione della *persistence logic* per la gestione dei trasferimenti nel microservizio **Order**. Questa struttura utilizza una mappa per memorizzare i trasferimenti e un mutex per garantire la sicurezza dei dati in caso di accesso concorrente.

Descrizione degli attributi della struttura:

- **`m sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente;
- **`transferMap map[string]Transfer`**: mappa che associa l'identificativo di un trasferimento[G] (**string**) all'oggetto `Transfer` corrispondente.

Descrizione dei metodi invocabili dalla struttura:

- **`NewTransferRepositoryImpl() *TransferRepositoryImpl`**: costruttore della struttura. Inizializza l'attributo `transferMap` come una mappa vuota e ritorna un puntatore alla struttura creata;
- **`GetTransfer(transferId string) (Transfer, error)`**: restituisce i dettagli di un trasferimento[G] specifico tramite il suo identificativo. Ritorna un errore `ErrTransferNotFound` se il trasferimento[G] non esiste;
- **`GetTransfers() []Transfer`**: restituisce una lista di tutti i trasferimenti registrati nel sistema;

- **`SetTransfer(transferId string, transfer Transfer) bool`**: aggiunge o aggiorna un trasferimento[G] nella mappa. Ritorna `true` se il trasferimento[G] esisteva già, `false` altrimenti;
- **`SetComplete(transferId string) error`**: segna un trasferimento[G] come completato. Ritorna un errore `ErrTransferNotFound` se il trasferimento[G] non esiste;
- **`IncrementLinkedStockUpdate(transferId string) error`**: incrementa il numero di aggiornamenti dello stock[G] associati a un trasferimento[G]. Ritorna un errore `ErrTransferNotFound` se il trasferimento[G] non esiste.

3.4.8.35 - TransferPersistenceAdapter

Adapter che mette in comunicazione la *business logic* con la *persistence logic* per la gestione dei trasferimenti.

Descrizione degli attributi della struttura:

- **`transferRepo ITransferRepository`**: rappresenta il *repository*[G] utilizzato per accedere e gestire i dati relativi ai trasferimenti.

Descrizione dei metodi invocabili dalla struttura:

- **`NewTransferPersistenceAdapter(transferRepo ITransferRepository) *TransferPersistenceAdapter`**: costruttore della struttura. Inizializza l'attributo `transferRepo` con il *repository*[G] fornito come parametro.
- **`SetComplete(transferId model.TransferID) error`**: segna un trasferimento[G] come completato. Prende come parametro l'identificativo del trasferimento[G] (`transferId`) e restituisce un errore in caso di fallimento.
- **`IncrementLinkedStockUpdate(transferId model.TransferID) error`**: incrementa il numero di aggiornamenti dello stock[G] associati a un trasferimento[G]. Prende come parametro l'identificativo del trasferimento[G] (`transferId`) e restituisce un errore in caso di fallimento.
- **`ApplyTransferUpdate(cmd port.ApplyTransferUpdateCmd) error`**: applica un aggiornamento di un trasferimento[G]. Prende come parametro un comando `ApplyTransferUpdateCmd` contenente i dettagli dell'aggiornamento e restituisce un errore in caso di fallimento.
- **`GetTransfer(transferId model.TransferID) (model.Transfer, error)`**: restituisce i dettagli di un trasferimento[G] specifico. Prende come parametro l'identificativo del trasferimento[G] (`transferId`) e restituisce un oggetto `Transfer` e un eventuale errore in caso di fallimento.
- **`GetAllTransfer() []Transfer`**: restituisce una lista di tutti i trasferimenti registrati nel sistema. Restituisce una slice di oggetti `Transfer`.
- **`repoTransferToModelTransfer(transfer model.Transfer) model.Transfer`**: converte un oggetto `Transfer` del *repository*[G] in un oggetto `Transfer` utilizzato nella business logic.
- **`repoTransfersToModelTransfers(transfers []model.Transfer) []model.Transfer`**: converte una lista di oggetti `Transfer` del *repository*[G] in una lista di oggetti `Transfer` utilizzati nella business logic.

3.4.8.36 - IStockRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* per la gestione dello stock[G] nel microservizio **Order**.

Descrizione dei metodi dell'interfaccia:

- **`GetStock(warehouseId string, goodId string) (int64, error)`**: il metodo deve permettere di ottenere la quantità di una merce specifica in un determinato magazzino. Prende come parametri l'identificativo del magazzino (`warehouseId`) e l'identificativo della merce (`goodId`). Restituisce la quantità della merce come `int64` e un eventuale errore in caso di fallimento;
- **`SetStock(warehouseId string, goodId string, stock int64) bool`**: il metodo deve permettere di impostare la quantità di una merce in un determinato magazzino. Prende come parametri l'identificativo del magazzino (`warehouseId`), l'identificativo della merce (`goodId`) e la quantità da impostare (`stock#super[G]`). Restituisce `true` se l'operazione è andata a buon fine, `false` altrimenti;
- **`AddStock(warehouseId string, goodId string, stock int64) (bool, error)`**: il metodo deve permettere di aggiungere una quantità specifica di una merce a un determinato magazzino. Prende come parametri l'identificativo del magazzino (`warehouseId`), l'identificativo della merce (`goodId`) e la quantità da aggiungere (`stock#super[G]`). Restituisce `true` se l'operazione è andata a buon fine, `false` altrimenti, e un eventuale errore in caso di fallimento;
- **`GetGlobalStock(goodId string) int64`**: il metodo deve permettere di ottenere la quantità globale di una merce presente in tutti i magazzini. Prende come parametro l'identificativo della merce (`goodId`) e restituisce la quantità globale come `int64`;
- **`GetWarehouses() []string`**: il metodo deve permettere di ottenere l'elenco di tutti i magazzini registrati nel sistema. Restituisce una slice di stringhe contenente gli identificativi dei magazzini.

3.4.8.37 - StockRepositoryImpl

La struttura `StockRepositoryImpl` rappresenta l'implementazione della *persistence logic* per la gestione dello stock[G] nel microservizio **Order**. Questa struttura utilizza una mappa per tenere traccia dello stock[G] di ciascun magazzino e una mappa globale per monitorare la quantità totale di ciascuna merce.

Descrizione degli attributi della struttura:

- **`m sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente;
- **`warehouseMap map[string]WarehouseStock`**: mappa che associa l'identificativo di un magazzino (**string**) a un oggetto `WarehouseStock`, contenente le informazioni sullo stock[G] del magazzino;
- **`globalStockMap map[string]int64`**: mappa che associa l'identificativo di una merce (**string**) alla quantità totale disponibile nei vari magazzini (**int64**).

Descrizione dei metodi invocabili dalla struttura:

- **`NewStockRepositoryImpl() *StockRepositoryImpl`**: costruttore della struttura. Inizializza le mappe `warehouseMap` e `globalStockMap` come mappe vuote e ritorna un puntatore alla struttura creata;
- **`GetStock(warehouseId string, goodId string) (int64, error)`**: restituisce la quantità di una merce specifica in un determinato magazzino. Ritorna un errore `ErrWarehouseNotFound` se il magazzino non esiste e `ErrGoodNotFound` se la merce non è presente nel magazzino;
- **`SetStock(warehouseId string, goodId string, stock int64) bool`**: imposta la quantità di una merce in un determinato magazzino. Se il magazzino non esiste, viene creato automaticamente. Ritorna `true` se la merce esisteva già, `false` altrimenti;
- **`AddStock(warehouseId string, goodId string, stock int64) (bool, error)`**: aggiunge una quantità specifica di una merce a un determinato magazzino. Ritorna `true` se la merce esisteva già, `false` altrimenti, e un errore `ErrWarehouseNotFound` o `ErrGoodNotFound` in caso di fallimento;
- **`GetGlobalStock(goodId string) int64`**: restituisce la quantità totale di una merce disponibile in tutti i magazzini. Se la merce non esiste, ritorna 0;
- **`GetWarehouses() []string`**: restituisce una lista degli identificativi di tutti i magazzini registrati nel sistema.

3.4.8.38 - StockPersistenceAdapter

Adapter che mette in comunicazione la *business logic* del microservizio **Order** con la *persistence logic* per la gestione dello stock[G].

Descrizione degli attributi della struttura:

- **`stockRepo IStockRepository`**: rappresenta il *repository*[G] utilizzato per accedere e gestire i dati relativi allo stock[G].

Descrizione dei metodi invocabili dalla struttura:

- **`NewStockPersistenceAdapter(stockRepo IStockRepository) *StockPersistenceAdapter`**: costruttore della struttura. Inizializza l'attributo `stockRepo` con il *repository*[G] fornito come parametro.
- **`ApplyStockUpdate(cmd port.ApplyStockUpdateCmd) error`**: applica un aggiornamento dello stock[G]. Per ogni merce inclusa nel comando, aggiorna la quantità nel magazzino specificato.
- **`GetStock(cmd port.GetStockCmd) (model.GoodStock, error)`**: restituisce la quantità di una merce specifica in un determinato magazzino. Se il magazzino non esiste, restituisce un errore `ErrStockNotFound`.
- **`GetGlobalStock(GoodID model.GoodID) model.GoodStock`**: restituisce la quantità globale di una merce presente in tutti i magazzini.
- **`GetWarehouses() []model.Warehouse`**: restituisce una lista di tutti i magazzini registrati nel sistema.

3.4.8.39 - IOrderRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* per la gestione degli ordini nel microservizio **Order**.

Descrizione dei metodi dell'interfaccia:

- **`GetOrder(orderId string) (Order, error)`**: il metodo deve permettere di ottenere i dettagli di un ordine[G] specifico, prendendo come parametro l'identificativo dell'ordine (`orderId`). Deve restituire un oggetto `Order` contenente i dettagli dell'ordine e un eventuale errore in caso di fallimento;
- **`GetOrders() []Order`**: il metodo deve permettere di ottenere una lista di tutti gli ordini registrati nel sistema. Deve restituire una slice di oggetti `Order`;
- **`SetOrder(orderId string, order Order) bool`**: il metodo deve permettere di impostare o aggiornare i dettagli di un ordine[G] specifico, prendendo come parametri l'identificativo dell'ordine (`orderId`) e un oggetto `Order` contenente i dettagli aggiornati. Deve restituire `true` se l'operazione è andata a buon fine, `false` altrimenti;
- **`AddCompletedWarehouse(orderId string, warehouseId string, goods map[string]int64) (Order, error)`**: il metodo deve permettere di aggiungere un magazzino completato a un ordine[G] specifico, aggiornando le quantità delle merci coinvolte. Deve restituire l'ordine aggiornato e un eventuale errore in caso di fallimento;
- **`SetComplete(orderId string) error`**: il metodo deve permettere di segnare un ordine[G] come completato, prendendo come parametro l'identificativo dell'ordine (`orderId`). Deve restituire un errore in caso di fallimento.

3.4.8.40 - OrderRepositoryImpl

Oggetto che implementa la *persistence logic* per la gestione degli ordini nel microservizio **Order**.

Descrizione degli attributi della struttura:

- **`m sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente;
- **`orderMap map[string]Order`**: mappa che associa l'identificativo di un ordine[G] (**string**) all'oggetto `Order` corrispondente.

Descrizione dei metodi invocabili dalla struttura:

- **`NewOrderRepositoryImpl() *OrderRepositoryImpl`**: costruttore della struttura. Inizializza l'attributo `orderMap` come una mappa vuota e ritorna un puntatore alla struttura creata;
- **`GetOrder(orderId string) (Order, error)`**: restituisce i dettagli di un ordine[G] specifico tramite il suo identificativo. Ritorna un errore `ErrOrderNotFound` se l'ordine non esiste;
- **`GetOrders() []Order`**: restituisce una lista di tutti gli ordini registrati nel sistema;
- **`SetOrder(orderId string, order Order) bool`**: aggiunge o aggiorna un ordine[G] nella mappa. Ritorna `true` se l'ordine esisteva già, `false` altrimenti;
- **`AddCompletedWarehouse(orderId string, warehouseId string, goods map[string]int64) (Order, error)`**: aggiunge un magazzino completato a un ordine[G] specifico, aggiornando le quantità delle merci coinvolte. Ritorna l'ordine aggiornato e un errore `ErrOrderNotFound` se l'ordine non esiste;
- **`SetComplete(orderId string) error`**: segna un ordine[G] come completato. Ritorna un errore `ErrOrderNotFound` se l'ordine non esiste.

3.4.8.41 - OrderPersistenceAdapter

Adapter che mette in comunicazione la *business logic* del microservizio **Order** con la *persistence logic* per la gestione degli ordini.

Descrizione degli attributi della struttura:

- **`orderRepo IOrderRepository`**: rappresenta il *repository*[G] utilizzato per accedere e gestire i dati relativi agli ordini.

Descrizione dei metodi invocabili dalla struttura:

- **`NewOrderPersistenceAdapter(orderRepo IOrderRepository) *OrderPersistenceAdapter`**: costruttore della struttura. Inizializza l'attributo `orderRepo` con il *repository*[G] fornito come parametro.
- **`SetCompletedWarehouse(cmd port.SetCompletedWarehouseCmd) (model.Order, error)`**: segna un magazzino come completato per un ordine[G] specifico. Aggrega le quantità delle merci coinvolte e aggiorna l'ordine nel *repository*[G]. Restituisce l'ordine aggiornato in formato `Order` e un eventuale errore in caso di fallimento.
- **`SetComplete(orderId model.OrderID) error`**: segna un ordine[G] come completato nel *repository*[G]. Restituisce un errore in caso di fallimento.
- **`ApplyOrderUpdate(cmd port.ApplyOrderUpdateCmd)`**: applica un aggiornamento a un ordine[G] esistente o crea un nuovo ordine[G] se non esiste. Aggiorna i dettagli dell'ordine, inclusi i magazzini, le merci e le prenotazioni, e salva l'ordine nel *repository*[G].
- **`GetOrder(orderId model.OrderID) (model.Order, error)`**: restituisce i dettagli di un ordine[G] specifico dal *repository*[G]. Converte l'ordine dal formato del *repository*[G] a `Order`. Restituisce un errore in caso di fallimento.
- **`GetAllOrder() []model.Order`**: restituisce una lista di tutti gli ordini registrati nel *repository*[G]. Converte gli ordini dal formato del *repository*[G] a `Order`.
- **`repoOrderToModelOrder(order model.Order) model.Order`**: funzione di utilità che converte un oggetto `Order` del *repository*[G] in un oggetto `Order` utilizzato nella business logic.
- **`repoOrdersToModelOrders(orders []model.Order) []mddel.Order`**: funzione di utilità che converte una lista di oggetti `Order` del *repository*[G] in una lista di oggetti `Order` utilizzati nella business logic.

3.4.8.42 - NatsStreamAdapter

Rappresenta un *Adapter* che mette in comunicazione la *business logic* con il sistema di messaggistica NATS[G] per gestire aggiornamenti di ordini, trasferimenti, contatti con i magazzini e richieste di prenotazione.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`**: rappresenta il broker di messaggistica NATS[G] utilizzato per pubblicare e ricevere messaggi.

Descrizione dei metodi invocabili dalla struttura:

- **`NewNatsStreamAdapter(broker *broker.NatsMessageBroker) *NatsStreamAdapter`**: costruttore della struttura. Inizializza l'attributo `broker` con il valore passato come parametro.

- **`SendOrderUpdate(ctx context.Context, cmd port.SendOrderUpdateCmd) (model.Order, error)`**: invia un aggiornamento di un ordine[G] al sistema di messaggistica NATS[G]. Restituisce un oggetto `Order` contenente i dettagli dell'ordine aggiornato e un eventuale errore in caso di fallimento.
- **`SendTransferUpdate(ctx context.Context, cmd port.SendTransferUpdateCmd) (model.Transfer, error)`**: invia un aggiornamento di un trasferimento[G] al sistema di messaggistica NATS[G]. Restituisce un oggetto `Transfer` contenente i dettagli del trasferimento[G] aggiornato e un eventuale errore in caso di fallimento.
- **`SendContactWarehouses(ctx context.Context, cmd port.SendContactWarehouseCmd) error`**: invia un comando per contattare i magazzini al sistema di messaggistica NATS[G]. Restituisce un errore in caso di fallimento.
- **`RequestReservation(ctx context.Context, cmd port.RequestReservationCmd) (port.RequestReservationResponse, error)`**: invia una richiesta di prenotazione di merci al sistema di messaggistica NATS[G]. Restituisce un oggetto `RequestReservationResponse` contenente l'identificativo della prenotazione creata e un eventuale errore in caso di fallimento.

3.4.8.43 - HealthCheckController

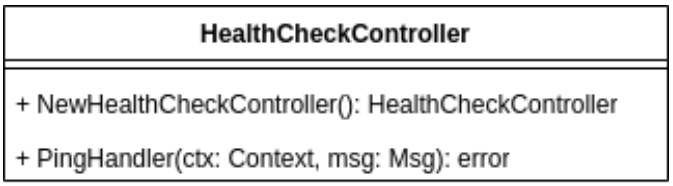

Figura 104: Order - HealthCheckController

La struttura `HealthCheckController` rappresenta l'*application logic* per la gestione delle richieste di controllo dello stato di salute del microservizio. È progettata per rispondere a richieste di tipo «ping» con una risposta «pong», indicando che il microservizio è attivo e funzionante.

Descrizione degli attributi della struttura:

Questa struttura non possiede attributi.

Descrizione dei metodi invocabili dalla struttura:

- **`NewHealthCheckController() *HealthCheckController`**: costruttore della struttura. Restituisce un'istanza di `HealthCheckController`.
- **`PingHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce le richieste di tipo «ping» ricevute tramite il sistema di messaggistica NATS[G]. Risponde con un messaggio «pong». Ritorna un errore in caso di fallimento durante la serializzazione della risposta o l'invio del messaggio di risposta.

3.4.9 - Catalog

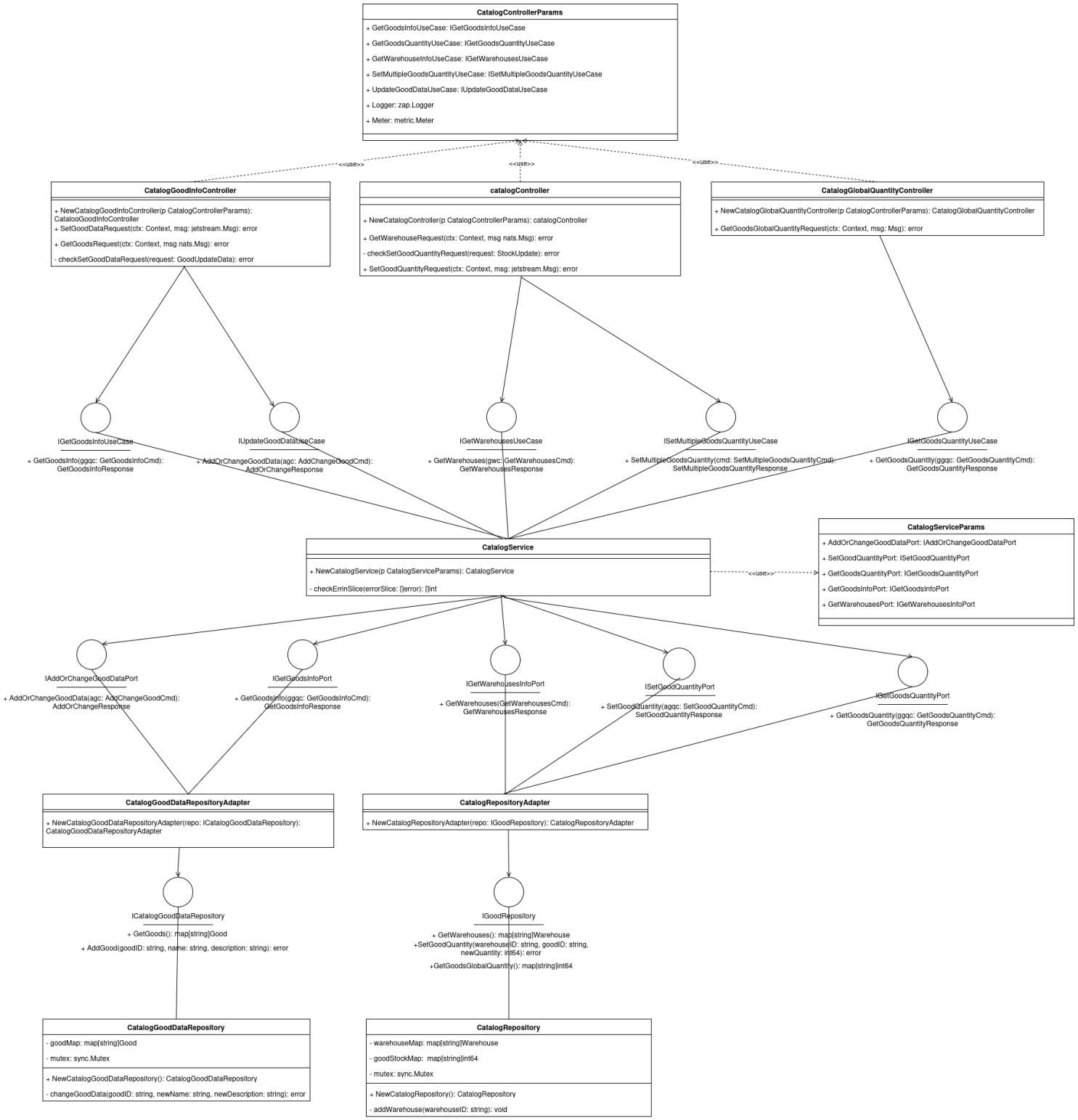

Figura 105: Componenti del microservizio Catalog

Il microservizio **Catalog** viene utilizzato per tenere traccia dell'inventario globale e della situazione di ciascun singolo magazzino.

Il microservizio tiene traccia dell'aggiunta e della modifica delle informazioni delle merci, prestando attenzione anche all'aggiunta di *stock*[G] delle merci stesse.

È formato da tre componenti principali:

- **CatalogController**, **CatalogGoodInfoController** e **CatalogGlobalQuantityController**, che rappresenta l'*application logic*;
- **CatalogService**, che rappresenta la *business logic*;
- **CatalogRepository** e **CatalogGoodDataRepository**, che rappresenta la *persistence logic*.

Le tre componenti, assieme agli oggetti eventualmente utilizzati saranno ora esposti.

3.4.9.1 - Oggetti comuni del microservizio

3.4.9.1.1 - Warehouse

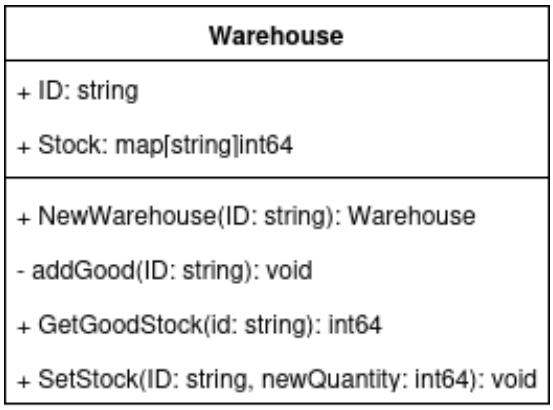

Figura 106: Catalog - Warehouse

Rappresenta un magazzino registrato nel Sistema.

Descrizione degli attributi della struttura:

- **`ID string`**, attributo di tipo **string** che rappresenta l'Id del magazzino;
- **`Stock map[string]int64`**, mappa che ha come chiave una **string** (identificativo della merce) e come valore un **int64** (la quantità della rispettiva merce nel presente magazzino)

Descrizione dei metodi invocabili dalla struttura:

- **`NewWarehouse(ID string) *Warehouse`**: rappresenta il costruttore della classe, prende una **string** come parametro per inizializzare l'id del magazzino;
- **`SetStock(ID string, newQuantity int64)`**: per modificare la quantità della merce con id pari al parametro **string** nel valore passato come parametro **int64**;
- **`addGood(ID string)`**: per aggiungere una merce nel magazzino, impostando il rispettivo id nel valore di tipo **string** passato come parametro.

3.4.9.1.2 - Good

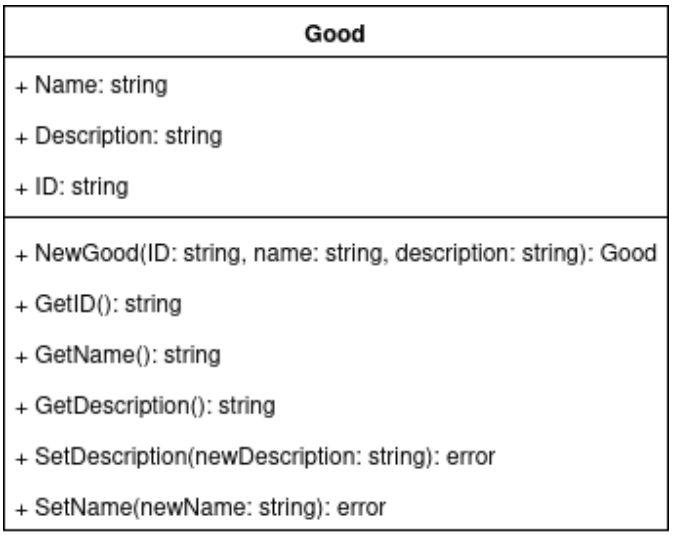

Figura 107: Catalog - Good

Rappresenta una merce registrata nel Sistema.

Descrizione degli attributi della struttura:

- **`Name string`**: attributo **string** che rappresenta il nome della merce;
- **`Description string`**: attributo **string** che rappresenta la descrizione della merce;
- **`ID string`**: attributo **string** che rappresenta l'id della merce

Descrizione dei metodi invocabili dalla struttura:

- **`NewGood(ID string, name string, description string) *Good`**: rappresenta il costruttore della struttura, restituisce un **Good**[G] inizializzato ai valori passati come parametro;
- **`GetID() string`**: restituisce l'id della merce;
- **`GetName() string`**: restituisce il nome della merce;
- **`GetDescription() string`**: restituisce la descrizione della merce;
- **`SetDescription(newDescription string) error`**: imposta la descrizione della merce al valore passato come parametro. Restituisce un errore se il parametro è una stringa vuota;
- **`SetName(newName string) error`**: imposta il nome della merce al valore passato come parametro. Restituisce un errore se il parametro è una stringa vuota.

3.4.9.1.3 - AddChangeGoodCmd

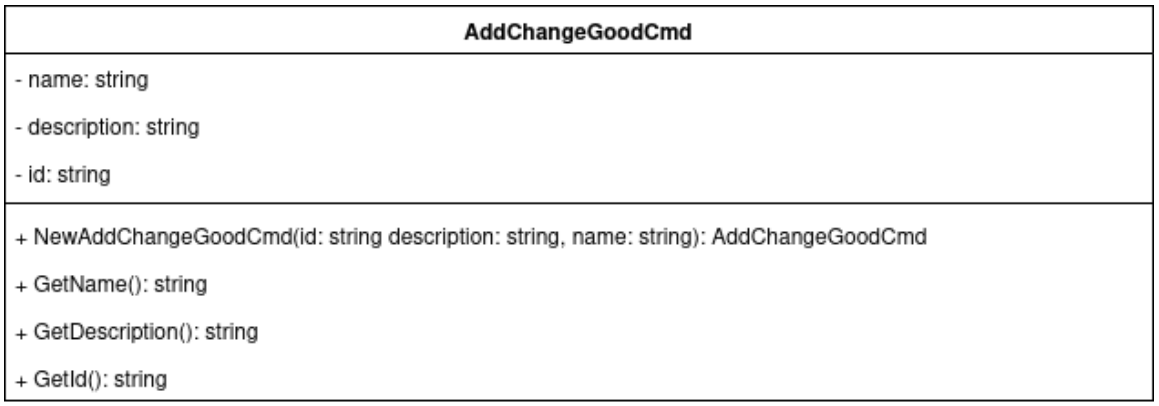

Figura 108: Catalog - AddChangeGoodCmd

Rappresenta il *Command* per aggiungere o modificare le informazioni di una merce.

Descrizione degli attributi della struttura:

- **`id string`**: rappresenta l'id della merce da aggiungere o modificare;
- **`name string`**: rappresenta il nuovo nome da assegnare alla merce in questione;
- **`description`**: rappresenta la nuova descrizione da assegnare alla merce in questione;

Descrizione dei metodi invocabili dalla struttura:

- **`NewAddChangeGoodCmd(id string, name string, description string) *AddChangeGoodCmd`**: rappresenta il costruttore della struttura, che viene inizializzata in base ai parametri richiesti (**id** per identificare l'id della merce interessata, **name** per indicarne il nome e **description** per rappresentarne la descrizione);
- **`GetId() string`**: ritorna l'id registrato nel *Command*;
- **`GetName() string`**: ritorna il nome registrato nel *Command*;
- **`GetDescription() string`**: ritorna la descrizione registrata nel *Command*;

3.4.9.1.4 - GetGoodsInfoCmd

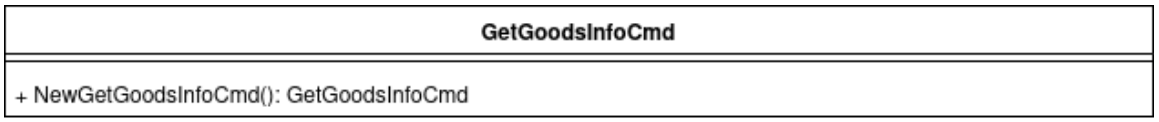

Figura 109: Catalog - GetGoodsInfoCmd

Rappresenta il *Command* per richiedere i dati delle merci registrate nel Sistema.

Descrizione degli attributi della struttura:

questa struttura non possiede attributi

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetGoodsInfoCmd() *GetGoodsInfoCmd`**: rappresenta il costruttore del *Command*.

3.4.9.1.5 - GetGoodsQuantityCmd

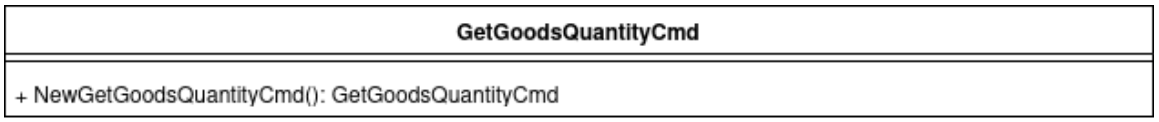

Figura 110: Catalog - GetGoodsQuantityCmd

Rappresenta il *Command* per richiedere la quantità globale delle merci registrate nel Sistema.

Descrizione degli attributi della struttura:

questa struttura non possiede attributi

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetGoodsQuantityCmd() *GetGoodsQuantityCmd`**: rappresenta il costruttore del *Command*.

3.4.9.1.6 - GetWarehousesCmd

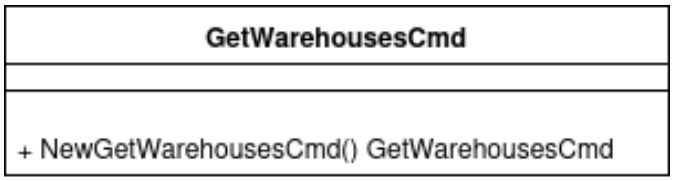

Figura 111: Catalog - GetWarehousesCmd

Rappresenta il *Command* per richiedere l'inventario dei magazzini registrati nel Sistema.

Descrizione degli attributi della struttura:

questa struttura non possiede attributi

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetWarehousesCmd() *GetWarehousesCmd`**: rappresenta il costruttore del *Command*.

3.4.9.1.7 - SetGoodQuantityCmd

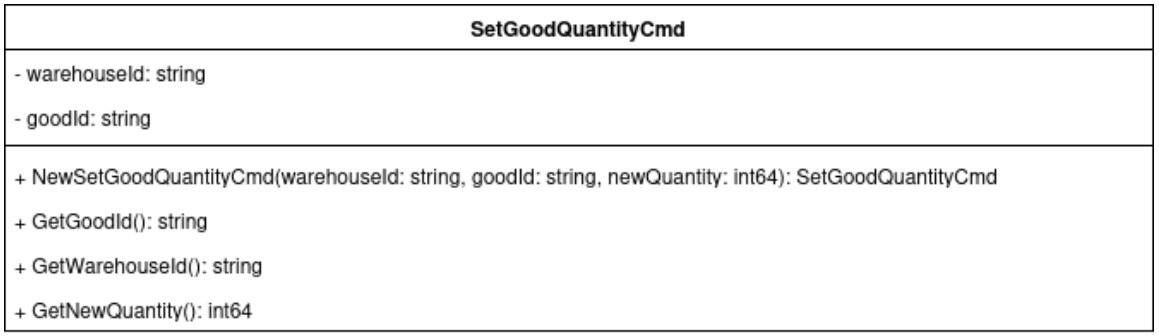

Figura 112: Catalog - SetGoodQuantityCmd

Rappresenta il *Command* per aggiornare la quantità di una merce in un magazzino e, conseguentemente, la quantità globale della merce stessa.

Descrizione degli attributi della struttura:

- **`warehouseID string`**: rappresenta l'id del magazzino dove la quantità della merce va modificata;

- **`goodID string`**: rappresenta l'id della merce la cui quantità va modificata;
- **`newQuantity int64`**: rappresenta la nuova quantità della merce in questione.

Descrizione dei metodi invocabili dalla struttura:

- **`NewSetGoodQuantityCmd(warehouseId string, goodId string, newQuantity int64) *SetGoodQuantityCmd`**: rappresenta il costruttore del *Command;*
- **`GetGoodId() string`**: permette di ottenere l'id della merce registrata nel *Command;*
- **`GetWarehouseId() string`**: permette di ottenere il magazzino registrato nel *Command;*
- **`GetNewQuantity() int64`**: permette di ottenere la quantità registrata nel *Command*.

3.4.9.1.8 - SetMultipleGoodsQuantityCmd

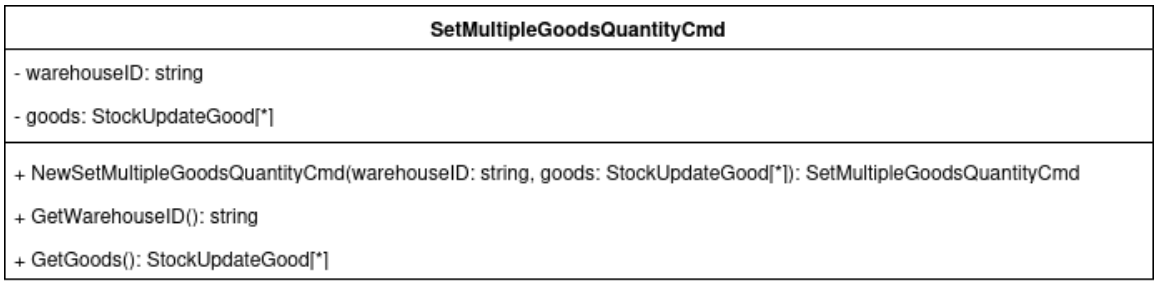

Figura 113: Catalog - SetMultipleGoodsQuantityCmd

Rappresenta il *Command* utilizzato per modificare le quantità di una serie di merci registrate nel magazzino.

Descrizione degli attributi della struttura:

- **`warehouseID string`**: rappresenta il magazzino interessato dalla modifica delle quantità delle merci;
- **`goods []stream.StockUpdateGood`**: rappresenta uno *slice* contenente le informazioni sulle merci da modificare.

Descrizione dei metodi invocabili dalla struttura:

- **`NewSetMultipleGoodsQuantityCmd(warehouseID string, goods []stream.StockUpdateGood) *SetMultipleGoodsQuantityCmd`**: è il costruttore del *Command*;
- **`GetGoods() []stream.StockUpdateGood`**: permette di ottenere lo *slice* delle merci da modificare;
- **`GetWarehouseID() string`**: permette di ottenere l'id del magazzino su cui effettuare le modifiche.

3.4.9.1.9 - AddOrChangeResponse

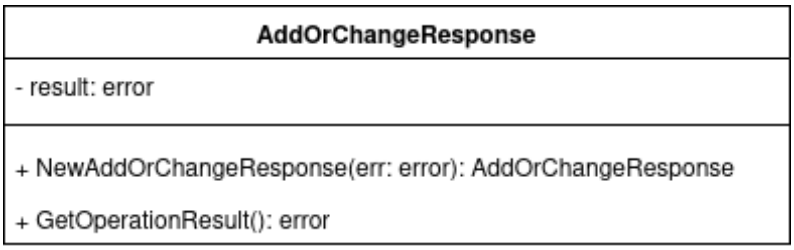

Figura 114: Catalog - AddOrChangeResponse

Rappresenta la Risposta alla richiesta di aggiunta o modifica informazioni di una merce.

Descrizione degli attributi della struttura:

- **`result error`**: viene qui memorizzato l'esito dell'operazione, se positivo (`nil`) o non.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAddOrChangeResponse(err error) *AddOrChangeResponse`**: rappresenta il costruttore della Risposta. **`result`** viene inizializzato fornendo al costruttore l'esito dell'operazione;
- **`GetOperationResult() string`**: permette di ottenere dalla Risposta l'esito dell'operazione.

3.4.9.1.10 - GetGoodsInfoResponse

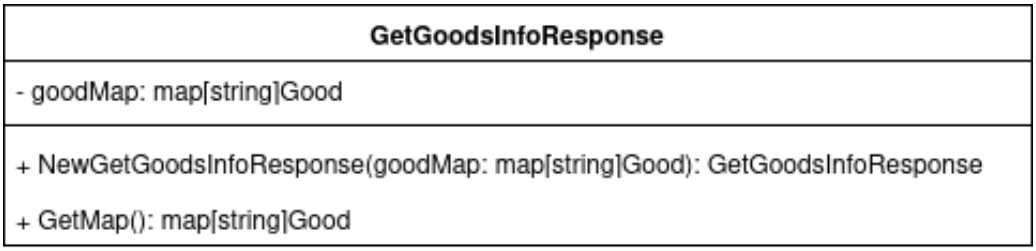

Figura 115: Catalog - GetGoodsInfoResponse

Rappresenta la Risposta all'operazione di richiesta informazioni su una merce.

Descrizione degli attributi della struttura:

- **`goodMap map[string]catalogCommon.Good`**: rappresenta la mappa delle merci ottenuta;

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetGoodsInfoResponse(goodMap map[string]catalogCommon.Good) *GetGoodsInfoResponse`**: rappresenta il costruttore della Risposta. La mappa viene inizializzata con quella passata come parametro al costruttore;
- **`GetMap() map[string]catalogCommon.Good`**: permette di ottenere la mappa memorizzata nella Risposta.

3.4.9.1.11 - GetGoodsQuantityResponse

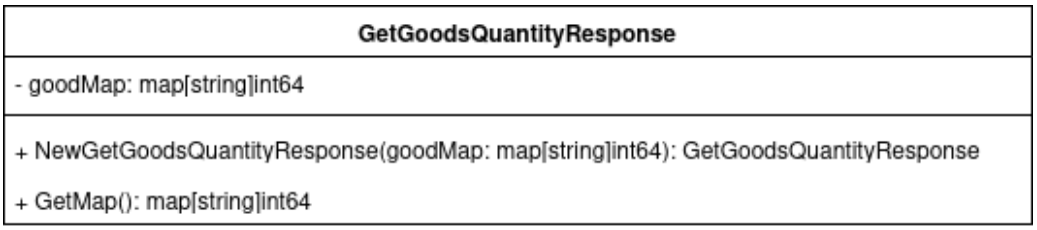

Figura 116: Catalog - GetGoodsQuantityResponse

Rappresenta la Risposta all'operazione di richiesta informazioni sulla quantità delle merci memorizzate nel Sistema.

Descrizione degli attributi della struttura:

- **`goodMap map[string]int64`**: mappa che contiene, per ogni id di merce memorizzata, la rispettiva quantità globale, memorizzata in **int64**;

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetGoodsQuantityResponse(goodMap map[string]int64) *GetGoodsQuantityResponse`**: rappresenta il costruttore della Risposta. La mappa viene inizializzata con quella passata come parametro nel costruttore;
- **`GetMap() map[string]int64`**: permette di ottenere la mappa memorizzata nella Risposta.

3.4.9.1.12 - GetWarehousesResponse

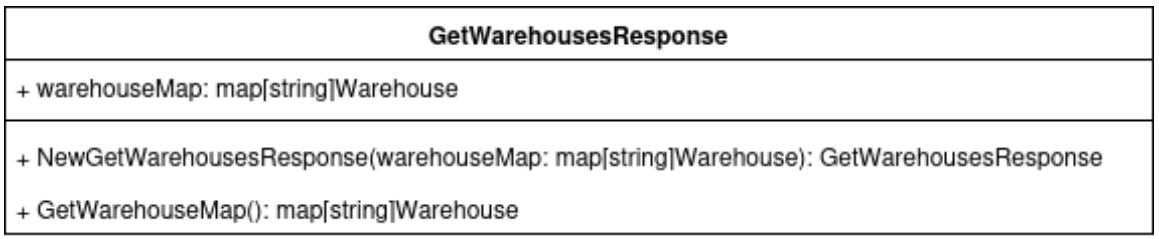

Figura 117: Catalog - GetWarehousesResponse

Rappresenta la Risposta all'operazione di richiesta informazioni sull'inventario dei magazzini memorizzati nel Sistema.

Descrizione degli attributi della struttura:

- **`warehouseMap map[string]catalogCommon.Warehouse`**: mappa che contiene, per ogni id di magazzino memorizzato nel Sistema, le informazioni sullo stesso.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetWarehousesResponse(warehouseMap map[string]catalogCommon.Warehouse) *GetWarehousesResponse`**: rappresenta il costruttore della Risposta. La mappa viene inizializzata con quella passata come parametro nel costruttore;
- **`GetWarehouseMap() map[string]catalogCommon.Warehouse`**: permette di ottenere la mappa memorizzata nella Risposta.

3.4.9.1.13 - SetGoodQuantityResponse

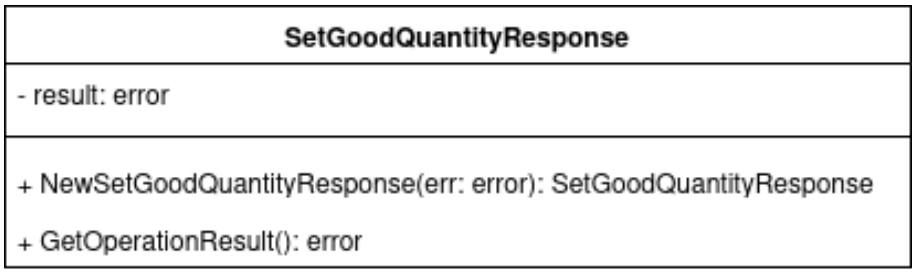

Figura 118: Catalog - SetGoodQuantityResponse

Rappresenta la Risposta alla richiesta di modifica quantità di una merce.

Descrizione degli attributi della struttura:

- **`result error`**: viene qui memorizzato l'esito dell'operazione, se positivo (`nil`) o non.

Descrizione dei metodi invocabili dalla struttura:

- **`NewSetGoodQuantityResponse(err error) *SetGoodQuantityResponse`**: rappresenta il costruttore della Risposta. **`result`** viene inizializzato fornendo al costruttore l'esito dell'operazione;
- **`GetOperationResult() string`**: permette di ottenere dalla Risposta l'esito dell'operazione.

3.4.9.1.14 - SetMultipleGoodsQuantityResponse

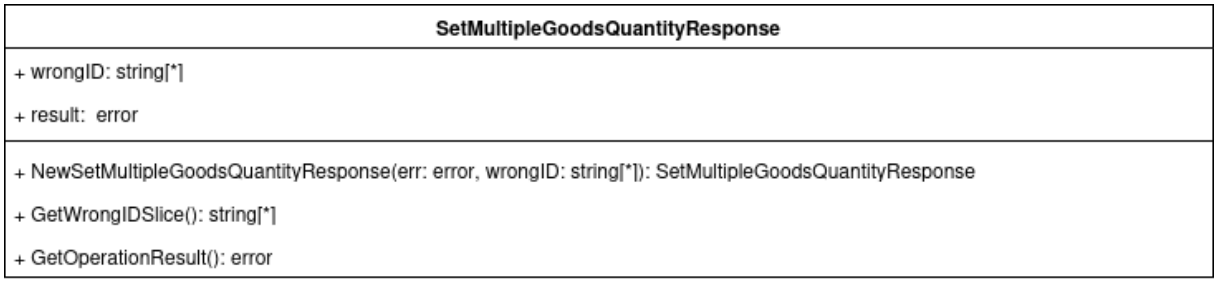

Figura 119: Catalog - SetMultipleGoodsQuantityResponse

Rappresenta la Risposta alla richiesta di modifica quantità di un insieme di merci.

Descrizione degli attributi della struttura:

- **`result error`**: viene qui memorizzato l'esito dell'operazione, se positivo (`nil`) o non.
- **`wrongID []string`**: rappresenta uno slice con gli id delle merci la cui modifica della quantità non è riuscita.

Descrizione dei metodi invocabili dalla struttura:

- **`NewSetMultipleGoodsQuantityResponse(err error, wrongID []string) *SetMultipleGoodsQuantityResponse`**: costruttore della Risposta. Gli attributi vengono inizializzati con i valori passati come parametri al costruttore;
- **`GetOperationResult() string`**: permette di ottenere dalla Risposta l'esito dell'operazione;
- **`GetWrongIDSlice() []string`**: permette di ottenere la *slice* degli id la cui modifica non è riuscita.

3.4.9.2 - CatalogGoodDataRepository

Questa struttura implementa l'interfaccia **ICatalogGoodDataRepository**, vedi la Sezione 3.4.9.4.

Descrizione degli attributi della struttura:

- **`goodMap map[string]*catalogCommon.Good`**: è una mappa che ha come chiave una **string** (l'identificatore della merce) e come valore un oggetto **Good**[G], rappresentante una merce;
- **`mutex sync.Mutex`**: variabile utilizzata per il corretto funzionamento di alcuni metodi. Si rimanda alla documentazione del linguaggio Go[G] per ulteriori informazioni.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogGoodDataRepository() *CatalogGoodDataRepository`**: rappresenta il costruttore della struttura. Non prende alcun parametro, inizializzando gli attributi a mappe vuote;
- **`GetGoods() map[string]catalogCommon.Good`**: restituisce la mappa delle merci internamente memorizzata;
- **`AddGood(goodID string, name string, description string) error`**: aggiunge una merce al Sistema con id **goodID**, nome **name** e descrizione **description**. Se la merce è già presente nel Sistema, chiama automaticamente la funzione `changeGoodData` per modificarne le informazioni. Ritorna sempre `nil`;
- **`changeGoodData(goodID string, newName string, newDescription string) error`**: cambia le informazioni della merce con id **goodID**, impostando il nome a **newName** e la descrizione a **newDescription**. Ritorna un errore se l'id della merce non è registrato.

3.4.9.3 - CatalogRepository

Questa struttura implementa l'interfaccia **IGoodRepository**, vedi la Sezione 3.4.9.5.

Descrizione degli attributi della struttura:

- **`warehouseMap map[string]*catalogCommon.Warehouse`**: è una mappa che ha come chiave una **string** (l'identificativo del magazzino) e come valore un oggetto **Warehouse**[G], rappresentante un magazzino;
- **`goodStockMap map[string]int64`**: è una mappa che ha come chiave una **string** (l'identificatore della merce) e come valore un **int64** (la quantità di quella merce tra tutti i magazzini)
- **`mutex sync.Mutex`**: variabile utilizzata per il corretto funzionamento di alcuni metodi. Si rimanda alla documentazione del linguaggio Go[G] per ulteriori informazioni

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogRepository() *CatalogRepository`**: rappresenta il costruttore della struttura. Non prende alcun parametro, inizializzando gli attributi a mappe vuote;
- **`GetGoodsGlobalQuantity() map[string]int64`**: restituisce la mappa della quantità globale delle merci;
- **`GetWarehouses() map[string]catalogCommon.Warehouse`**: restituisce la mappa dei magazzini riconosciuti dal Sistema;
- **`SetGoodQuantity(warehouseID string, goodID string, newQuantity int64) error`**: imposta la quantità della merce con id **goodID** del magazzino con id **warehouseID** alla

quantità memorizzata nel parametro **newQuantity**. In caso la merce sia nuova, questa viene automaticamente aggiunta, ma senza nome e descrizione. Ritorna sempre **`nil`**;

- **`addWarehouse(warehouseID string)`**: aggiunge magazzino al sistema con id pari a **warehouseID**. Questa operazione è effettuata automaticamente quando si cerca di aggiunge *stock*[G] ad un magazzino non ancora registrato;

3.4.9.4 - ICatalogGoodDataRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* del microservizio *Catalog* che gestisce i dati delle merci.

Descrizione dei metodi dell'interfaccia:

- **`GetGoods() map[string]catalogCommon.Good`**: il metodo deve dare possibilità di ottenere i dati delle merci registrate nel Sistema;
- **`AddGood(goodID string, name string, description string) error`**: il metodo deve dare la possibilità di aggiungere una merce al Sistema.

3.4.9.5 - IGoodRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* del microservizio *Catalog* che gestisce i magazzini e le quantità di merce.

Descrizione dei metodi dell'interfaccia:

- **`GetGoodsGlobalQuantity() map[string]int64`**: il metodo deve dare la possibilità di ottenere la quantità globale delle merci nel Sistema;
- **`SetGoodQuantity(warehouseID string, goodID string, newQuantity int64) error`**: il metodo deve dare la possibilità di impostare la quantità di una merce in un magazzino e, conseguentemente, la quantità globale;
- **`GetWarehouses() map[string]catalogCommon.Warehouse`**: il metodo deve dare la possibilità di ottenere i magazzini registrati nel Sistema.

3.4.9.6 - IAddOrChangeGoodDataPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di voler aggiungere o modificare i dati di una merce.

Descrizione dei metodi dell'interfaccia:

- **`AddOrChangeGoodData(agc *servicecmd.AddChangeGoodCmd) *serviceresponse.AddOrChangeResponse`**: il metodo deve dare la possibilità di aggiungere e/o modificare i dati di una merce.

3.4.9.7 - IGetGoodsInfoPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere i dati delle merci registrate nel Sistema.

Descrizione dei metodi dell'interfaccia:

- **`GetGoodsInfo(ggqc *servicecmd.GetGoodsInfoCmd) *serviceresponse.GetGoodsInfoResponse`**: il metodo deve permettere di richiedere i dati sulle merci registrate nel Sistema e di ottenerle in risposta.

3.4.9.8 - IGetGoodsQuantityPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere le informazioni sulla quantità delle merci registrate nel Sistema.

Descrizione dei metodi dell'interfaccia:

- **`GetGoodsQuantity(ggqc *servicecmd.GetGoodsQuantityCmd) *serviceresponse.GetGoodsQuantityResponse`**: il metodo deve permettere di richiedere i dati sulla quantità delle merci registrate nel Sistema e di ottenerle in risposta.

3.4.9.9 - IGetWarehousesInfoPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere le informazioni sull'inventario dei magazzini registrati nel Sistema.

Descrizione dei metodi dell'interfaccia:

- **`GetWarehouses(*servicecmd.GetWarehousesCmd) *serviceresponse.GetWarehousesResponse`**: il metodo deve permettere di richiedere le informazioni sull'inventario dei magazzini registrati nel Sistema e ottenere tali informazioni in risposta.

3.4.9.10 - ISetGoodQuantityPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di impostare la quantità di una merce.

Descrizione dei metodi dell'interfaccia:

- **`SetGoodQuantity(agqc *servicecmd.SetGoodQuantityCmd) *serviceresponse.SetGoodQuantityResponse`**: il metodo deve permettere di modificare la quantità di una merce.

3.4.9.11 - CatalogGoodDataRepositoryAdapter

Adapter che mette in comunicazione la *business logic* di catalog con la *persistence logic* relativa alla gestione dati merci merci dello stesso.

Implementa le seguenti interfacce (porte):

- **IAddOrChangeGoodDataPort**, Sezione 3.4.9.6;
- **IGetGoodsInfoPort**, Sezione 3.4.9.7.

Descrizione degli attributi della struttura:

- **`repo persistence.ICatalogGoodDataRepository`**: l'*Adapter* possiede un attributo alla struttura rappresentante la *persistence logic* di Catalog relativa alla gestione dati delle merci. Per le informazioni riguardo IGoodRepository vedere la Sezione 3.4.9.4.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogGoodDataRepositoryAdapter(repo persistence.ICatalogGoodDataRepository) *CatalogGoodDataRepositoryAdapter`**: costruttore dell'*Adapter*. Inizializza l'attributo `repo` con quello passato come parametro al costruttore;
- **`AddOrChangeGoodData(agc *servicecmd.AddChangeGoodCmd) *serviceresponse.AddOrChangeResponse`**: converte il *Command* per l'aggiunta o modifica dati merce in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;

- **`GetGoodsInfo(ggqc *servicecmd.GetGoodsInfoCmd) *serviceresponse.GetGoodsInfoResponse`**: converte il *Command* per ottenere le informazioni sulle varie merci registrate nel Sistema in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata.

3.4.9.12 - CatalogRepositoryAdapter

Adapter che mette in comunicazione la *business logic* di catalog con la *persistence logic* relativa alla gestione magazzini e quantità merci dello stesso.

Implementa le seguenti interfacce (porte):

- **IGetGoodsQuantityPort**, Sezione 3.4.9.8;
- **IGetWarehousesInfoPort**, Sezione 3.4.9.9;
- **ISetGoodQuantityPort**, Sezione 3.4.9.10.

Descrizione degli attributi della struttura:

- **`repo persistence.IGoodRepository`**: l'*Adapter* possiede un attributo alla struttura rappresentante la *persistence logic* di Catalog relativa alla gestione magazzini e quantità merci. Per le informazioni riguardo IGoodRepository vedere la Sezione 3.4.9.5.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogRepositoryAdapter(repo persistence.IGoodRepository) *CatalogRepositoryAdapter`**: costruttore dell'*Adapter*. Inizializza l'attributo `repo` con quello passato come parametro al costruttore;
- **`SetGoodQuantity(agqc *servicecmd.SetGoodQuantityCmd) *serviceresponse.SetGoodQuantityResponse`**:converte il *Command* per la modifica della quantità di una merce in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;
- **`GetGoodsQuantity(ggqc *servicecmd.GetGoodsQuantityCmd) *serviceresponse.GetGoodsQuantityResponse`**: converte il *Command* per ottenere la quantità delle varie merci registrate nel Sistema in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;
- **`GetWarehouses(*servicecmd.GetWarehousesCmd) *serviceresponse.GetWarehousesResponse`**: converte il *Command* per ottenere le informazioni sui magazzini registrati nel Sistema in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata.

3.4.9.13 - IService

Interfaccia che descrive i metodi che devono essere implementati da una struttura che si propone di soddisfare la *business logic* del microservizio Catalog.

Descrizione dei metodi dell'interfaccia:

- **`AddOrChangeGoodData(agc *servicecmd.AddChangeGoodCmd) *serviceresponse.AddOrChangeResponse`**: il metodo deve permettere di aggiungere o modificare le informazioni di una merce;
- **`SetMultipleGoodsQuantity(cmd *servicecmd.SetMultipleGoodsQuantityCmd) *serviceresponse.SetMultipleGoodsQuantityResponse`**: il metodo deve permettere di aggiornare la quantità di un gruppo di merci;

- **`GetGoodsQuantity(ggqc *servicecmd.GetGoodsQuantityCmd) *serviceresponse.GetGoodsQuantityResponse`**: il metodo deve permettere di richiedere la quantità delle merci registrate nel Sistema e ottenerne la risposta;
- **`GetGoodsInfo(ggqc *servicecmd.GetGoodsInfoCmd) *serviceresponse.GetGoodsInfoResponse`**: il metodo deve permettere di richiedere le informazioni sulle merci memorizzate nel sistema e ottenerne la risposta;
- **`GetWarehouses(gwc *servicecmd.GetWarehousesCmd) *serviceresponse.GetWarehousesResponse`**: il metodo deve permettere di richiedere le informazione sull'inventario dei magazzini memorizzati nel Sistema e ottenerne la risposta.

3.4.9.14 - IGetGoodsInfoUseCase

Rappresenta l'interfaccia che permette, all'*application logic* di comunicare alla *business logic* la volontà di ottenere informazioni sulle merci memorizzate.

Descrizione dei metodi dell'interfaccia:

- **`GetGoodsInfo(ggqc *servicecmd.GetGoodsInfoCmd) *serviceresponse.GetGoodsInfoResponse`**: il metodo deve permettere di richiedere le informazioni sulle merci memorizzate nel sistema e ottenerne la risposta.

3.4.9.15 - IGetGoodsQuantityUseCase

Rappresenta l'interfaccia che permette, all'*application logic* di comunicare alla *business logic* la volontà di ottenere informazioni sulla quantità delle merci memorizzate.

Descrizione dei metodi dell'interfaccia:

- **`GetGoodsQuantity(ggqc *servicecmd.GetGoodsQuantityCmd) *serviceresponse.GetGoodsQuantityResponse`**: il metodo deve permettere di richiedere la quantità delle merci registrate nel Sistema e ottenerne la risposta.

3.4.9.16 - IGetWarehousesUseCase

Rappresenta l'interfaccia che permette, all'*application logic* di comunicare alla *business logic* la volontà di ottenere informazioni sull'inventario dei magazzini memorizzati nel Sistema.

Descrizione dei metodi dell'interfaccia:

- **`GetWarehouses(gwc *servicecmd.GetWarehousesCmd) *serviceresponse.GetWarehousesResponse`**: il metodo deve permettere di richiedere le informazione sull'inventario dei magazzini memorizzati nel Sistema e ottenerne la risposta.

3.4.9.17 - ISetMultipleGoodsQuantityUseCase

Rappresenta l'interfaccia che permette, all'*application logic* di comunicare alla *business logic* la volontà di impostare la quantità di varie merci.

Descrizione dei metodi dell'interfaccia:

- **`SetMultipleGoodsQuantity(cmd *servicecmd.SetMultipleGoodsQuantityCmd) *serviceresponse.SetMultipleGoodsQuantityResponse`**: il metodo deve permettere di aggiornare la quantità di un gruppo di merci.

3.4.9.18 - IUpdateGoodDataUseCase

Rappresenta l'interfaccia che permette, all'*application logic* di comunicare alla *business logic* la volontà di modificare le informazioni di una merce nel Sistema.

Descrizione dei metodi dell'interfaccia:

- **`AddOrChangeGoodData(agc *servicecmd.AddChangeGoodCmd) *serviceresponse.AddOrChangeResponse`**: il metodo deve permettere di aggiungere o modificare le informazioni di una merce.

3.4.9.19 - CatalogService

Si occupa di gestire la *business logic* del microservizio *Catalog* e implementa, per questo motivo, **IService** (Sezione 3.4.9.13).

Implementa le seguenti interfacce (*Use Case*):

- **IGetGoodsInfoUseCase**, Sezione 3.4.9.14;
- **IGetGoodsQuantityUseCase**, Sezione 3.4.9.15;
- **IGetWarehousesUseCase**, Sezione 3.4.9.16;
- **ISetMultipleGoodsQuantityUseCase**, Sezione 3.4.9.17;
- **IUpdateGoodDataUseCase**, Sezione 3.4.9.18.

Descrizione degli attributi della struttura:

- **`addOrChangeGoodDataPort serviceportout.IAddOrChangeGoodDataPort`**: vedere la descrizione alla Sezione 3.4.9.6;
- **`setGoodQuantityPort serviceportout.ISetGoodQuantityPort`**: vedere la descrizione alla Sezione 3.4.9.10;
- **`getGoodsQuantityPort serviceportout.IGetGoodsQuantityPort`**: vedere la descrizione alla Sezione 3.4.9.8;
- **`getGoodsInfoPort serviceportout.IGetGoodsInfoPort`**: vedere la descrizione alla Sezione 3.4.9.7;
- **`getWarehousesPort serviceportout.IGetWarehousesInfoPort`**: vedere la descrizione alla Sezione 3.4.9.9.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogService(p CatalogServiceParams) *CatalogService`**: Costruttore della struttura. Le porte (*Use Case*) devono essere fornite come parametri al costruttore e, per farlo, si utilizza la struttura **`CatalogServiceParams`**, struttura con i medesimi attributi di CatalogService con l'istruzione `fx.in` per permettere al *framework* **fx** di fornire automaticamente le dipendenze necessarie;
- **`AddOrChangeGoodData(agc *servicecmd.AddChangeGoodCmd) *serviceresponse.AddOrChangeResponse`**: prende un *Command* per la richiesta di aggiunta o cambiamento informazioni di una merce e attiva la porta adibita allo scopo per svolgere la richiesta. Ritorna quindi l'esito dell'operazione;
- **`SetMultipleGoodsQuantity(cmd *servicecmd.SetMultipleGoodsQuantityCmd)`**: prende un *Command* per la richiesta di modifica quantità di un gruppo di merce e trasmette la richiesta per ciascuna di tali merci alla porta adibita allo scopo. Richiama la funzione **`checkErrinSlice`** per controllare l'esito di ciascuna delle operazioni;

- **`checkErrinSlice(errorSlice []error) []int`**: controlla la *slice* passata come parametro per comprendere se un'operazione di modifica quantità merce non è andata a buon fine. Ritorna una *slice* con le posizioni nella *slice* contenente le merci da modificare in cui è stato riscontrato un esito negativo;
- **`GetGoodsQuantity(ggqc *servicecmd.GetGoodsQuantityCmd) *serviceresponse.GetGoodsQuantityResponse`**: prende un *Command* per la richiesta delle informazioni sulla quantità delle merci memorizzate nel Sistema e ne chiede esecuzione mediante l'apposita porta. Ritorna quindi la risposta;
- **`GetGoodsInfo(ggqc *servicecmd.GetGoodsInfoCmd) *serviceresponse.GetGoodsInfoResponse`**: prende un *Command* per la richiesta delle informazioni sulle merci memorizzate nel Sistema. Inoltra la richiesta alla porta opportuna e ritorna quindi la risposta;
- **`GetWarehouses(gwc *servicecmd.GetWarehousesCmd) *serviceresponse.GetWarehousesResponse`**: prende un *Command* per la richiesta delle informazioni sull'inventario dei magazzini memorizzati nel Sistema. Procede ad inoltrare la richiesta sulla porta opportuna e ritorna quindi la risposta.

3.4.9.20 - CatalogGlobalQuantityController

Si occupa di gestire l'*application logic* del microservizio Catalog in merito all'ottenimento delle quantità globalmente disponibili.

Descrizione degli attributi della struttura:

- **`GetGoodsQuantityUseCase serviceportin.IGetGoodsQuantityUseCase`**: la descrizione è disponibile alla Sezione 3.4.9.15;

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogGlobalQuantityController(p CatalogControllerParams) *CatalogGlobalQuantityController`**: costruttore della struttura. Gli attributi della struttura vengono inizializzati con i valori passati mediante la struttura `CatalogControllerParams`, che ha gli attributi necessari e quanto utile per la telemetria;
- **`GetGoodsGlobalQuantityRequest(ctx context.Context, msg *nats.Msg) error`**: metodo utilizzato per recuperare le richieste di ottenimento informazioni sulla quantità globalmente disponibile delle merci memorizzate nel Sistema. La richiesta arriva direttamente mediante un messaggio su **NATS**[G]. Ritorna un errore in caso l'operazione non venga completata correttamente;

3.4.9.21 - CatalogGoodInfoController

Si occupa di gestire l'*application logic* del microservizio Catalog in merito alla gestione dati delle merci.

Descrizione degli attributi della struttura:

- **`GetGoodsInfoUseCase serviceportin.IGetGoodsInfoUseCase`**: la descrizione è disponibile alla Sezione 3.4.9.14;
- **`UpdateGoodDataUseCase serviceportin.IUpdateGoodDataUseCase`**: la descrizione è disponibile alla Sezione 3.4.9.18.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogGoodInfoController(p CatalogControllerParams) *CatalogGoodInfoController`**: costruttore della struttura. Gli attributi della struttura vengono inizializzati con i valori passati mediante la struttura `CatalogControllerParams`, che ha gli attributi necessari e quanto utile per la telemetria;
- **`SetGoodDataRequest(ctx context.Context, msg jetstream.Msg)`**: metodo utilizzato per recuperare le richieste di aggiunta merce o modifica informazioni su una merce. La richiesta arriva direttamente mediante un messaggio su **NATS JetStream**. Utilizza il metodo `checkSetGoodDataRequest` per verificare se l'elaborazione della richiesta è sensata. Ritorna un errore in caso l'operazione non venga completata correttamente;
- **`GetGoodsRequest(ctx context.Context, msg *nats.Msg) error`**: metodo utilizzato per recuperare le richieste di ottenimento informazioni sulle merci presenti nel Sistema. La richiesta arriva direttamente mediante un messaggio su **NATS**[G]. Ritorna un errore in caso l'operazione non venga completata correttamente;

3.4.9.22 - CatalogController

Si occupa di gestire l'*application logic* del microservizio Catalog in merito alla gestione magazzini e quantità merce.

Descrizione degli attributi della struttura:

- **`GetWarehouseInfoUseCase serviceportin.IGetWarehousesUseCase`**: la descrizione è disponibile alla Sezione 3.4.9.16;
- **`SetMultipleGoodsQuantityUseCase serviceportin.ISetMultipleGoodsQuantityUseCase`**: la descrizione è disponibile alla Sezione 3.4.9.17;
- **`checkSetGoodDataRequest(request *stream.GoodUpdateData) error`**: controlla le richieste di aggiornamento dati o aggiunta merce. Ritorna un errore se la richiesta non è valida;

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogController(p CatalogControllerParams) *catalogController`**: costruttore della struttura. Gli attributi della struttura vengono inizializzati con i valori passati mediante la struttura `CatalogControllerParams`, che ha gli attributi necessari e quanto utile per la telemetria;;
- **`GetWarehouseRequest(ctx context.Context, msg *nats.Msg) error`**: metodo utilizzato per recuperare le richieste di ottenimento informazioni sui magazzini presenti nel Sistema e il loro inventario. La richiesta arriva direttamente mediante un messaggio su **NATS**[G]. Ritorna un errore in caso l'operazione non venga completata correttamente;
- **`SetGoodQuantityRequest(ctx context.Context, msg jetstream.Msg) error`**: metodo utilizzato per recuperare i messaggi relativi a richieste di aggiornamento della quantità di una merce. La richiesta arriva direttamente mediante un messaggio su **NATS JetStream**. Utilizza il metodo `checkSetGoodQuantityRequest` per verificare se l'elaborazione della richiesta è sensata. Ritorna un errore in caso l'operazione non venga completata correttamente;
- **`checkSetGoodQuantityRequest(request *stream.StockUpdate) error`**: controlla le richieste di aggiornamento quantità di una merce. Ritorna un errore se la richiesta non è valida.

3.4.10 - Warehouse

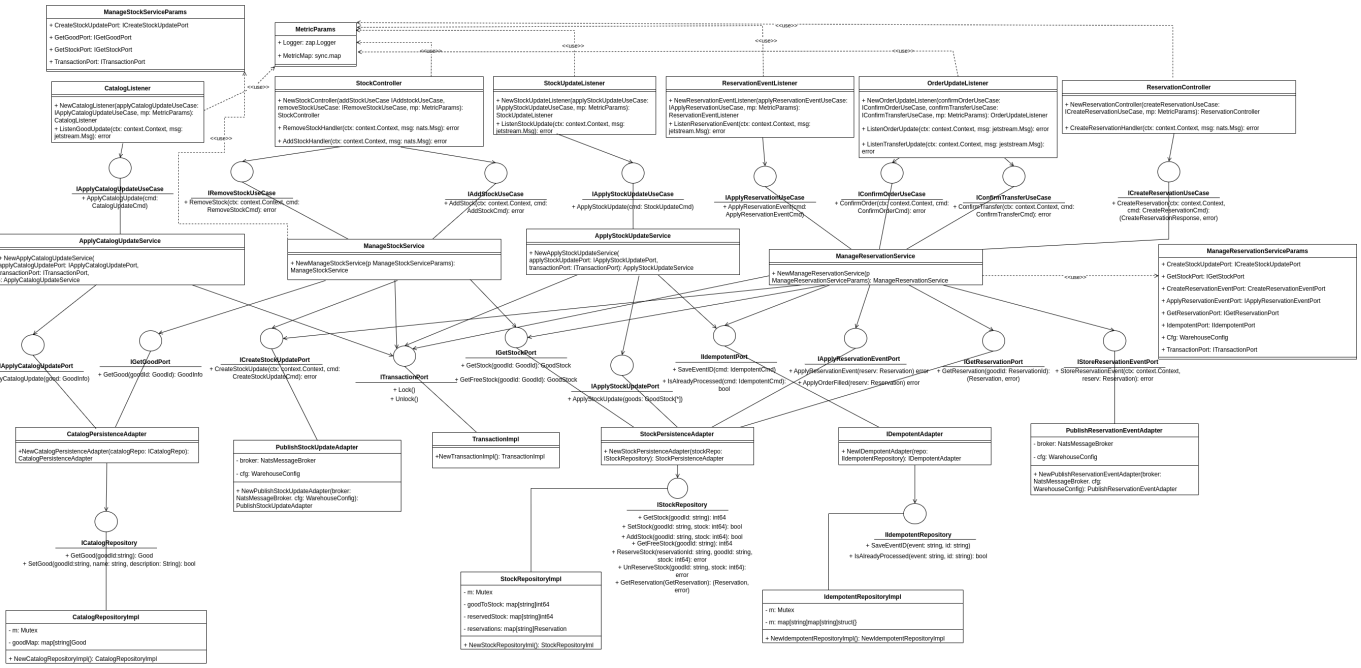

Figura 120: Componenti del microservizio Warehouse

Il microservizio **Warehouse**[G] viene utilizzato per gestire un determinato magazzino, tenendo traccia delle merci presenti al suo interno e della quantità di ciascuna di esse.

Il microservizio tiene traccia dell'aggiunta e della modifica delle informazioni delle merci.

Può funzionare anche in caso di mancanza di connessione con gli altri microservizi in quanto mantiene uno stato interno aggiornato all'ultima versione prima della disconnessione.

È formato da tre sotto-aree di componenti principali:

- I **Controller** e **Listener**, che rappresentano l'*application logic*
- I **Service**, che rappresentano la *business logic*;
- I **Repository**[G], che rappresentano la *persistence logic*.

Gli oggetti utilizzati per implementare queste componenti saranno ora esposti.

3.4.10.1 - Oggetti comuni

3.4.10.1.1 - Good (Repo)

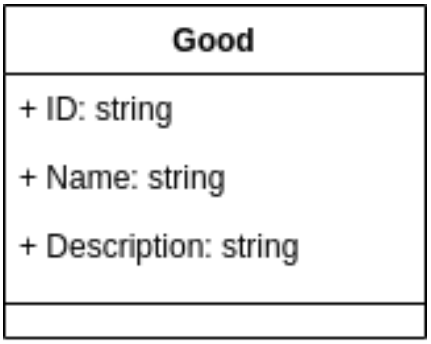

Figura 121: Warehouse - Good (Repo)

Rappresenta una merce registrata nel magazzino. Vieni utilizzato dall'interfaccia ICatalogRepository, vedi Sezione 3.4.10.4, per memorizzare le informazioni delle merci.

Descrizione degli attributi della struttura:

- **`ID string`**: attributo di tipo **string** che rappresenta l'Id della merce;
- **`Name string`**: attributo di tipo **string** che rappresenta il nome della merce;
- **`Description string`**: attributo di tipo **string** che rappresenta la descrizione della merce.

3.4.10.1.2 - Reservation (Repo)

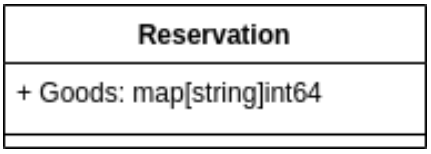

Figura 122: Warehouse - Reservation (Repo)

Rappresenta una prenotazione di merci nel magazzino. Viene utilizzata dalla *persistence logic* e dall'interfaccia **IStockRepository** (Sezione 3.4.10.2).

Descrizione degli attributi della struttura:

- **`Goods map[string]int64`**: rappresenta una mappa che associa l'identificativo di una merce (**string**) alla quantità prenotata (**int64**).

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.10.1.3 - Reservation

Figura 123: Warehouse - Reservation

Rappresenta una prenotazione di merci nel magazzino.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco della prenotazione;
- **`Goods []ReservationGood`**: rappresenta una lista di oggetti `ReservationGood` che contengono le informazioni sulle merci coinvolte nella prenotazione.

3.4.10.1.4 - ReservationID

Rappresenta un identificativo univoco per una prenotazione.

Descrizione degli attributi della struttura:

- **`string`**: rappresenta l'identificativo univoco della prenotazione.

3.4.10.1.5 - ReservationGood

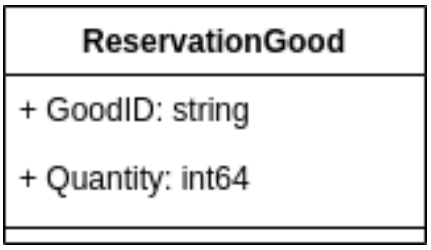

Figura 124: Warehouse - ReservationGood

Rappresenta una merce coinvolta in una prenotazione.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nella prenotazione.

3.4.10.1.6 - CreateReservationCmd

Figura 125: Warehouse - CreateReservationCmd

Rappresenta il comando utilizzato per creare una prenotazione.

Descrizione degli attributi della struttura:

- **`Goods []ReservationGood`**: rappresenta una lista di oggetti `ReservationGood` che contengono le informazioni sulle merci da prenotare.

3.4.10.1.7 - CreateReservationResponse

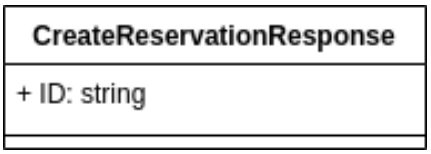

Figura 126: Warehouse - CreateReservationResponse

Rappresenta la risposta alla richiesta di creazione di una prenotazione.

Descrizione degli attributi della struttura:

- **`ReservationID string`**: rappresenta l'identificativo univoco della prenotazione creata.

3.4.10.1.8 - ConfirmTransferCmd

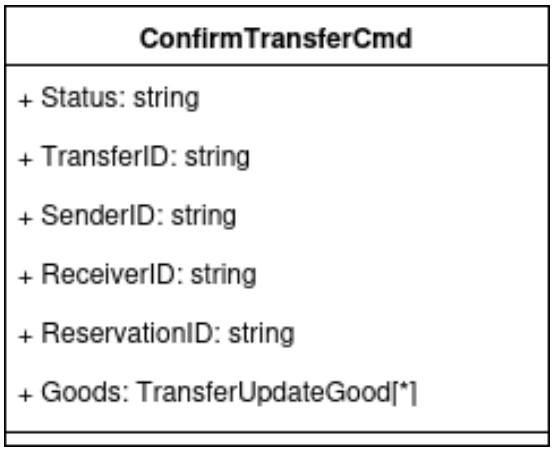

Figura 127: Warehouse - ConfirmTransferCmd

Rappresenta il comando utilizzato per confermare un trasferimento[G].

Descrizione degli attributi della struttura:

- **`TransferID string`**: rappresenta l'identificativo univoco del trasferimento[G] da confermare;
- **`SenderID string`**: rappresenta l'identificativo del magazzino mittente del trasferimento[G]
- **`ReceiverID string`**: rappresenta l'identificativo del magazzino destinatario del trasferimento[G]
- **`Status string`**: rappresenta lo stato del trasferimento[G] da aggiornare;
- **`Goods []TransferUpdateGood`**: rappresenta una lista di oggetti `TransferUpdateGood` che contengono le informazioni sulle merci coinvolte nel trasferimento[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata al trasferimento[G].

3.4.10.1.9 - TransferUpdateGood

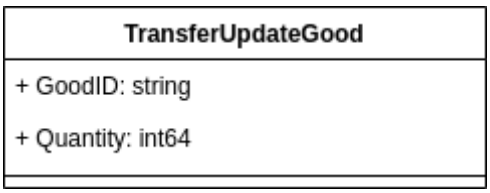

Figura 128: Warehouse - TransferUpdateGood

Rappresenta una merce coinvolta in un aggiornamento di un trasferimento[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nel trasferimento[G].

3.4.10.1.10 - ConfirmOrderCmd

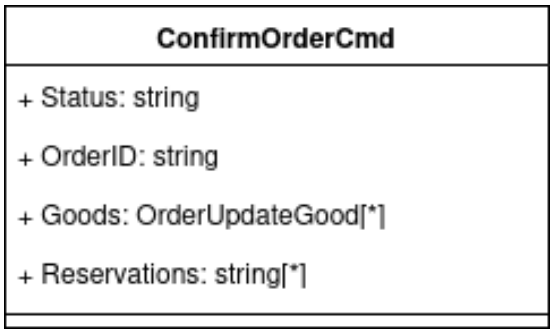

Figura 129: Warehouse - ConfirmOrderCmd

Rappresenta il comando utilizzato per confermare un ordine[G].

Descrizione degli attributi della struttura:

- **`OrderID string`**: rappresenta l'identificativo univoco dell'ordine da confermare;
- **`Status string`**: rappresenta lo stato dell'ordine da aggiornare;
- **`Goods []OrderUpdateGood`**: rappresenta una lista di oggetti `OrderUpdateGood` che contengono le informazioni sulle merci coinvolte nell'ordine;
- **`Reservations []string`**: rappresenta una lista di identificativi delle prenotazioni associate all'ordine.

3.4.10.1.11 - OrderUpdateGood

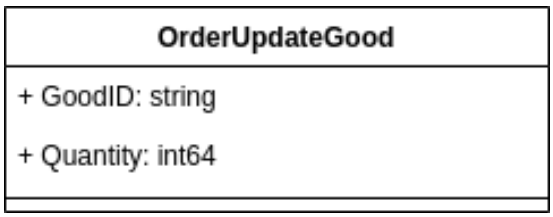

Figura 130: Warehouse - OrderUpdateGood

Rappresenta una merce coinvolta in un aggiornamento di un ordine[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nell'ordine.

3.4.10.1.12 - CatalogUpdateCmd

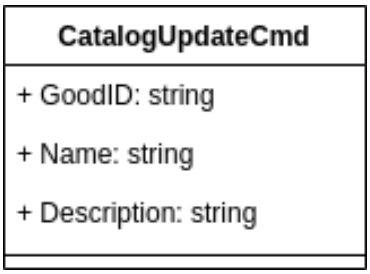

Figura 131: Warehouse - CatalogUpdateCmd

Rappresenta il comando per aggiornare le informazioni di un catalogo nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce da aggiornare;
- **`Name string`**: rappresenta il nuovo nome della merce;
- **`Description string`**: rappresenta la nuova descrizione della merce.

3.4.10.1.13 - StockUpdateCmd

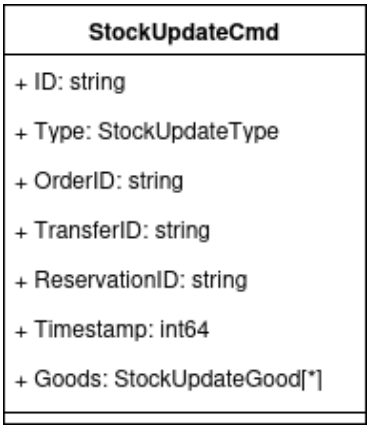

Figura 132: Warehouse - StockUpdateCmd

Rappresenta il comando per aggiornare lo stock[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco del comando di aggiornamento dello *stock*[G]
- **`Type StockUpdateType`**: rappresenta il tipo di aggiornamento dello *stock*[G]. Può assumere i seguenti valori:
  - **`add`**: per aggiungere stock[G]
  - **`remove`**: per rimuovere stock[G]
  - **`order`**: per aggiornamenti legati a ordini;
  - **`transfer`**: per aggiornamenti legati a trasferimenti;
- **`Goods []StockUpdateGood`**: rappresenta una lista di oggetti `StockUpdateGood` che contengono le informazioni sulle merci aggiornate;
- **`OrderID string`**: rappresenta l'identificativo dell'ordine associato all'aggiornamento dello *stock*[G]
- **`TransferID string`**: rappresenta l'identificativo del trasferimento[G] associato all'aggiornamento dello *stock*[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata all'aggiornamento dello *stock*[G]
- **`Timestamp int64`**: rappresenta il *timestamp* dell'aggiornamento dello *stock*[G].

3.4.10.1.14 - StockUpdateGood

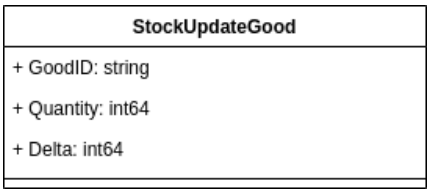

Figura 133: Warehouse - StockUpdateGood

Rappresenta una merce aggiornata nel comando di aggiornamento dello stock[G].

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce aggiornata;
- **`Quantity int64`**: rappresenta la nuova quantità della merce aggiornata;
- **`Delta int64`**: rappresenta la differenza di quantità della merce rispetto all'ultimo stato.

3.4.10.1.15 - StockUpdateType

Rappresenta il tipo di aggiornamento dello stock[G]. È un tipo stringa con i seguenti valori possibili:

- **`add`**: per aggiungere stock[G]
- **`remove`**: per rimuovere stock[G]
- **`order`**: per aggiornamenti legati a ordini;
- **`transfer`**: per aggiornamenti legati a trasferimenti.

3.4.10.1.16 - AddStockCmd

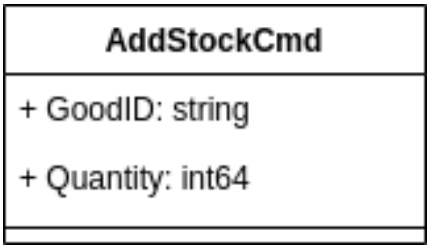

Figura 134: Warehouse - AddStockCmd

Questo *Command* viene utilizzato per rappresentare la richiesta di aggiunta di stock[G], e viene utilizzato dal caso d'uso[G] Sezione 3.4.10.26.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'id della merce a cui aggiungere stock[G]
- **`Quantity int64`**: rappresenta la quantità di stock[G] da aggiungere alla merce.

3.4.10.1.17 - IdempotentCmd

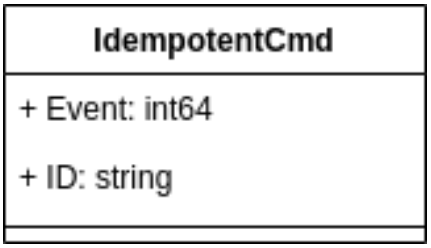

Figura 135: Warehouse - IdempotentCmd

Rappresenta il comando utilizzato per identificare un evento in modo univoco e gestire l'idempotenza.

Descrizione degli attributi della struttura:

- **`Event string`**: rappresenta il nome o il tipo dell'evento da identificare;
- **`ID string`**: rappresenta l'identificativo univoco dell'evento.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non ha metodi invocabili.

3.4.10.1.18 - RemoveStockCmd

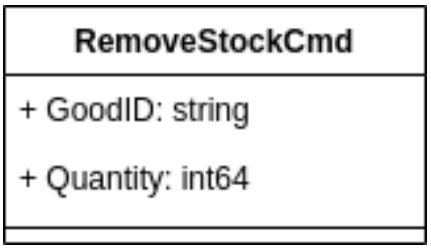

Figura 136: Warehouse - RemoveStockCmd

Questo *Command* viene utilizzato per rappresentare la richiesta di rimozione di stock[G], e viene utilizzato dal caso d'uso[G] Sezione 3.4.10.25.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'id della merce a cui rimuovere stock[G]
- **`Quantity int64`**: rappresenta la quantità di stock[G] da rimuovere dalla merce.

3.4.10.1.19 - GoodID

Rappresenta un identificativo univoco per una merce.

Descrizione del tipo:

- **`string`**: il tipo `GoodID` è un alias per il tipo `string`. Viene utilizzato per identificare in modo univoco una merce all'interno del sistema.

3.4.10.1.20 - GoodInfo

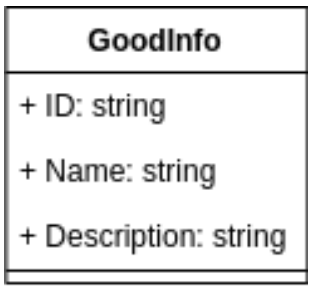

Figura 137: Warehouse - GoodInfo

Questa classe è utilizzata nella *business logic*. Rappresenta una merce con le sue informazioni.

Descrizione degli attributi della struttura:

- **`ID string`**: attributo di tipo **string** che rappresenta l'Id della merce;
- **`Name string`**: attributo di tipo **string** che rappresenta il nome della merce;
- **`Description string`**: attributo di tipo **string** che rappresenta la descrizione della merce.

3.4.10.1.21 - GoodStock

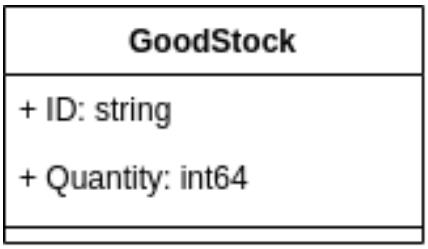

Figura 138: Warehouse - GoodStock

Questa classe è utilizzata nella *business logic*. Rappresenta una merce con la sua quantità presente nel magazzino.

Descrizione degli attributi della struttura:

- **`ID string`**: attributo di tipo **string** che rappresenta l'Id della merce;
- **`Quantity int64`**: attributo di tipo **int64** che rappresenta la quantità della merce nel magazzino.

3.4.10.1.22 - ReservationGood

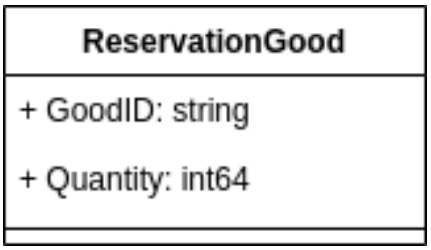

Figura 139: Warehouse - ReservationGood

Rappresenta una merce coinvolta in un evento di prenotazione.

Descrizione degli attributi della struttura:

- **`GoodID string`**: rappresenta l'identificativo della merce;
- **`Quantity int64`**: rappresenta la quantità della merce coinvolta nell'evento di prenotazione.

3.4.10.1.23 - ApplyReservationEventCmd

Figura 140: Warehouse - ApplyReservationEventCmd

Rappresenta il comando utilizzato per applicare un evento di prenotazione.

Descrizione degli attributi della struttura:

- **`ID string`**: rappresenta l'identificativo univoco dell'evento di prenotazione;
- **`Goods []ReservationGood`**: rappresenta una lista di oggetti `ReservationGood` che contengono le informazioni sulle merci coinvolte nell'evento di prenotazione.

3.4.10.1.24 - CreateStockUpdateCmd

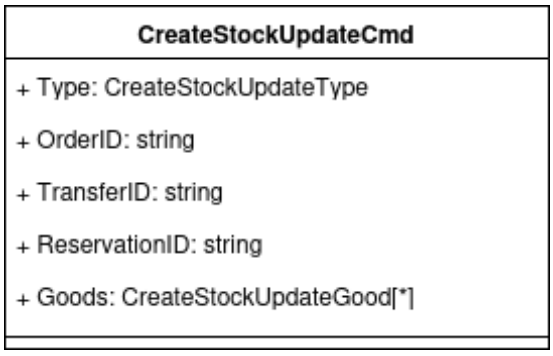

Figura 141: Warehouse - CreateStockUpdateCmd

Rappresenta il *Command* per creare un aggiornamento dello stock[G].

Descrizione degli attributi della struttura:

- **`Type CreateStockUpdateType`**: rappresenta il tipo di aggiornamento dello stock[G]. Può assumere i seguenti valori:
  - **`add`**: per aggiungere stock[G]
  - **`remove`**: per rimuovere stock[G]
  - **`order`**: per aggiornamenti legati a ordini;
  - **`transfer`**: per aggiornamenti legati a trasferimenti;
- **`Goods []CreateStockUpdateGood`**: rappresenta una lista di oggetti `CreateStockUpdateGood` che contengono le quantità delle merci aggiornate;
- **`OrderID string`**: rappresenta l'identificativo dell'ordine associato all'aggiornamento dello stock[G]
- **`TransferID string`**: rappresenta l'identificativo del trasferimento[G] associato all'aggiornamento dello stock[G]
- **`ReservationID string`**: rappresenta l'identificativo della prenotazione associata all'aggiornamento dello stock[G].

3.4.10.1.25 - CreateStockUpdateGood

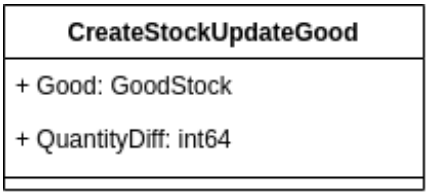

Figura 142: Warehouse - CreateStockUpdateGood

Rappresenta una classe che viene utilizzata dal *Command* per creare un aggiornamento dello stock[G].

Descrizione degli attributi della struttura:

- **`Good GoodStock`**: attributo di tipo `GoodStock` che rappresenta la merce aggiornata;
- **`QuantityDiff int64`**: attributo di tipo **int64** che rappresenta la differenza di quantità della merce rispetto all'ultimo stato.

3.4.10.2 - IStockRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* degli stock[G] per il microservizio *Warehouse*[G].

Descrizione dei metodi dell'interfaccia:

- **`GetStock(goodId string) int64`**: il metodo deve dare possibilità di ottenere la quantità di una merce presente nel magazzino;
- **`SetStock(goodId string, stock int64)`**: il metodo deve dare la possibilità di impostare la quantità di una merce nel magazzino;
- **`AddStock(goodId string, stock int64)`**: il metodo deve dare la possibilità di aggiungere una quantità di una merce ad un magazzino;
- **`GetFreeStock(goodId string) int64`**: il metodo deve dare la possibilità di ottenere la quantità di stock[G] disponibile per una merce nel magazzino;
- **`ReserveStock(reservationId string, goodId string, stock int64) error`**: il metodo deve dare la possibilità di riservare una quantità specifica di stock[G] per una merce, associandola a un identificativo di prenotazione;
- **`UnReserveStock(goodId string, stock int64) error`**: il metodo deve dare la possibilità di annullare una prenotazione di stock[G] per una merce, liberando la quantità riservata;
- **`GetReservation(reservationId string) (Reservation, error)`**: il metodo deve dare la possibilità di ottenere i dettagli di una prenotazione specifica tramite il suo identificativo.

3.4.10.3 - StockRepositoryImpl

Questa struttura implementa l'interfaccia **IStockRepository**, vedi la Sezione 3.4.10.2.

Descrizione degli attributi della struttura:

- **`m sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente;
- **`goodToStock map[string]int64`**: è una mappa che ha come chiave una **string** (l'identificativo della merce) e come valore un intero, rappresentante la quantità di stock[G] di quella merce;
- **`reservedStock map[string]int64`**: è una mappa che tiene traccia della quantità riservata di ogni merce, con l'identificativo della merce come chiave e la quantità riservata come valore;
- **`reservations map[string]Reservation`**: è una mappa che associa un identificativo di prenotazione a un oggetto `Reservation`, contenente i dettagli della prenotazione.

Descrizione dei metodi invocabili dalla struttura:

- **`NewStockRepositoryImpl() *StockRepositoryImpl`**: rappresenta il costruttore della struttura. Inizializza le mappe `goodToStock`, `reservedStock` e `reservations` come mappe vuote;
- **`GetStock(goodId string) int64`**: restituisce la quantità di una merce presente nel magazzino. Se la merce non esiste, ritorna 0;
- **`SetStock(goodId string, stock int64) bool`**: imposta la quantità di una merce nel magazzino. Ritorna `true` se la merce esisteva già, `false` altrimenti;

- **`AddStock(goodId string, stock int64) bool`**: aggiunge una quantità di una merce al magazzino. Ritorna `true` se la merce esisteva già, `false` altrimenti;
- **`ReserveStock(reservationId string, goodId string, stock int64) error`**: riserva una quantità specifica di una merce. Ritorna un errore se non c'è abbastanza stock[G] disponibile;
- **`UnReserveStock(goodId string, stock int64) error`**: annulla una prenotazione di stock[G] per una merce. Ritorna un errore se la quantità da annullare supera quella riservata;
- **`GetFreeStock(goodId string) int64`**: restituisce la quantità di stock[G] disponibile per una merce, calcolata come la differenza tra lo stock[G] totale e quello riservato;
- **`GetReservation(reservationId string) (Reservation, error)`**: restituisce i dettagli di una prenotazione specifica tramite il suo identificativo. Ritorna un errore se la prenotazione non esiste.

3.4.10.4 - ICatalogRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la *persistence logic* delle informazioni del catalogo per il microservizio *Warehouse*[G].

Descrizione dei metodi dell'interfaccia:

- **`GetGood(goodId string) *Good`**: il metodo deve dare la possibilità di ottenere i dati di una merce registrata nel magazzino tramite il suo ID;
- **`SetGood(goodId string, name string, description string) bool`**: il metodo deve dare la possibilità di aggiungere o modificare una merce nel magazzino.

3.4.10.5 - CatalogRepositoryImpl

Questa struttura implementa l'interfaccia **ICatalogRepository**, vedi la Sezione 3.4.10.4.

Descrizione degli attributi della struttura:

- **`goodMap map[string]Good`**: è una mappa che ha come chiave una **string** (l'identificatore della merce) e come valore un oggetto **Good**[G],rappresentante una merce.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogRepositoryImpl() *CatalogRepositoryImpl`**: rappresenta il costruttore della struttura. Non prende alcun parametro, inizializzando gli attributi a mappe vuote;
- **`GetGood(goodId string) *Good`**: restituisce i dati di una merce registrata nel magazzino tramite il suo ID;
- **`SetGood(goodId string, name string, description string) bool`**: aggiunge o modifica una merce nel magazzino con l'ID, nome e descrizione forniti.

3.4.10.6 - ICreateStockUpdatePort

Rappresenta la porta che consente alla *business logic* di comunicare con l'esterno per creare un aggiornamento dello stockRappresenta l'interfaccia generica di un oggetto che implementa la *Persistence Logic* degli stock[G] per il microservizio *Warehouse*[G].

Descrizione dei metodi dell'interfaccia:

- **`CreateStockUpdate(ctx: Context, CreateStockUpdateCmd) error`**: il metodo deve permettere di creare un aggiornamento dello stock[G].

3.4.10.7 - IStoreReservationEventPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di memorizzare un evento di prenotazione.

Descrizione dei metodi dell'interfaccia:

- **`StoreReservationEvent(ctx context.Context, reservation model.Reservation) error`**: il metodo deve permettere di memorizzare un evento di prenotazione, prendendo come parametri il contesto e un oggetto di tipo `Reservation`. Deve restituire un errore in caso di fallimento.

3.4.10.8 - IApplyReservationEventPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un evento di prenotazione o di applicare gli ordini ricevuti con stato `Filled`.

Descrizione dei metodi dell'interfaccia:

- **`ApplyReservationEvent(model.Reservation) error`**: il metodo deve permettere di applicare un evento di prenotazione, prendendo come parametro un oggetto di tipo `Reservation`. Deve restituire un errore in caso di fallimento;
- **`ApplyOrderFilled(model.Reservation) error`**: il metodo deve permettere di applicare gli ordini ricevuti con stato `Filled`, prendendo come parametro un oggetto di tipo `Reservation`. Deve restituire un errore in caso di fallimento.

3.4.10.9 - IApplyReservationUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di applicare un evento di prenotazione.

Descrizione dei metodi dell'interfaccia:

- **`ApplyReservationEvent(ApplyReservationEventCmd) error`**: il metodo deve permettere di applicare un evento di prenotazione, prendendo come parametro un oggetto di tipo `ApplyReservationEventCmd`. Deve restituire un errore in caso di fallimento.

3.4.10.10 - PublishReservationEventAdapter

Adapter che mette in comunicazione la *business logic* del microservizio **Warehouse**[G] con il sistema di messaggistica per inviare un modello di prenotazione trasformato in evento.

Implementa le seguenti interfacce (porte):

- **IStoreReservationEventPort**, Sezione 3.4.10.7.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`**: rappresenta il broker di messaggistica NATS[G] utilizzato per pubblicare i messaggi;
- **`warehouseCfg *config.WarehouseConfig`**: rappresenta la configurazione del magazzino.

Descrizione dei metodi invocabili dalla struttura:

- **`NewPublishReservationEventAdapter(broker *broker.NatsMessageBroker, warehouseCfg *config.WarehouseConfig) *PublishReservationEventAdapter`**: costruttore dell'adapter. Inizializza gli attributi `broker` e `warehouseCfg` con i valori passati come parametri;

- **`StoreReservationEvent(ctx context.Context, reservation model.Reservation) error`**: prende un modello di prenotazione, lo trasforma in un evento e lo invia al sistema di messaggistica NATS[G]. Ritorna un errore in caso di fallimento.

3.4.10.11 - PublishStockUpdateAdapter

Questa struttura implementa l'interfaccia **ICreateStockUpdatePort**, vedi la Sezione 3.4.10.6.

Adapter che mette in comunicazione la *business logic* con il sistema di messaggistica per pubblicare gli aggiornamenti dello stockRappresenta l'interfaccia generica di un oggetto che implementa la *Persistence Logic* degli stock[G] per il microservizio *Warehouse*[G].

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`**: rappresenta il broker di messaggistica NATS[G] utilizzato per pubblicare i messaggi;
- **`cfg *config.WarehouseConfig`**: rappresenta la configurazione del magazzino.

Descrizione dei metodi invocabili dalla struttura:

- **`NewPublishStockUpdateAdapter(broker *broker.NatsMessageBroker, *config.WarehouseConfig) *PublishStockUpdateAdapter`**: costruttore dell'adapter. Inizializza gli attributi `broker` e `cfg` con i valori passati come parametri.
- **`CreateStockUpdate(ctx context.Context, cmd port.CreateStockUpdateCmd) error`**: pubblica un aggiornamento dello stock[G] utilizzando il broker di messaggistica NATS[G].

3.4.10.12 - IApplyStockUpdatePort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento dello stock[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyStockUpdate(goods []model.GoodStock) error`**: il metodo deve permettere di applicare un aggiornamento dello stock[G].

3.4.10.13 - IGetStockPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere informazioni sulla quantità di una merce presente nel magazzino.

Descrizione dei metodi dell'interfaccia:

- **`GetStock(goodId model.GoodID) model.GoodStock`**: il metodo deve permettere di ottenere la quantità totale di una merce presente nel magazzino, prendendo come parametro l'identificativo della merce (`goodId`) e restituendo un oggetto di tipo `GoodStock`.
- **`GetFreeStock(goodId model.GoodID) model.GoodStock`**: il metodo deve permettere di ottenere la quantità libera di una merce presente nel magazzino, ovvero la quantità non riservata. Prende come parametro l'identificativo della merce (`goodId`) e restituisce un oggetto di tipo `GoodStock`.

3.4.10.14 - IGetReservationPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere i dettagli di una prenotazione specifica tramite il suo identificativo.

Descrizione dei metodi dell'interfaccia:

- **`GetReservation(reservationId model.ReservationID) (model.Reservation, error)`**: il metodo deve permettere di ottenere i dettagli di una prenotazione specifica, prendendo come parametro l'identificativo della prenotazione (`reservationId`) e restituendo un oggetto di tipo `Reservation` e un eventuale errore in caso di fallimento.

3.4.10.15 - ITransactionPort

Rappresenta l'interfaccia che consente alla *business logic* di gestire operazioni transazionali, fornendo metodi per bloccare e sbloccare risorse condivise.

Descrizione dei metodi dell'interfaccia:

- **`Lock()`**: il metodo deve permettere di bloccare una risorsa condivisa, garantendo che solo una transazione possa accedervi alla volta.
- **`Unlock()`**: il metodo deve permettere di sbloccare una risorsa condivisa, consentendo ad altre transazioni di accedervi.

3.4.10.16 - TransactionImpl

La struttura `TransactionImpl` rappresenta un'implementazione dell'interfaccia `ITransactionPort` per la gestione delle operazioni transazionali nel microservizio **Warehouse**[G]. Utilizza un mutex per garantire l'accesso esclusivo alle risorse condivise.

Descrizione degli attributi della struttura:

- **`m sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente.

Descrizione dei metodi invocabili dalla struttura:

- **`NewTransactionImpl() *TransactionImpl`**: costruttore della struttura. Inizializza l'attributo `m` con un nuovo mutex e restituisce un'istanza di `TransactionImpl`.
- **`Lock()`**: blocca il mutex, garantendo che solo una transazione possa accedere alla risorsa condivisa alla volta.
- **`Unlock()`**: sblocca il mutex, consentendo ad altre transazioni di accedere alla risorsa condivisa.

3.4.10.17 - IIdempotentPort

Rappresenta la porta che consente alla *business logic* di gestire operazioni idempotenti, assicurandosi che un evento non venga elaborato più di una volta.

Descrizione dei metodi dell'interfaccia:

- **`SaveEventID(IdempotentCmd)`**: il metodo deve permettere di salvare l'identificativo di un evento per garantire che non venga elaborato più volte;
- **`IsAlreadyProcessed(IdempotentCmd) bool`**: il metodo deve permettere di verificare se un evento è già stato elaborato, restituendo se l'evento è già stato processato.

3.4.10.18 - IIdempotentRepository

Rappresenta l'interfaccia generica di un oggetto che implementa la gestione dell'idempotenza per il microservizio *Warehouse*[G].

Descrizione dei metodi dell'interfaccia:

- **`SaveEventID(event string, id string)`**: il metodo deve permettere di salvare l'identificativo di un evento per garantire che non venga elaborato più volte. Prende come parametri il nome o tipo dell'evento (`event`) e il suo identificativo univoco (`id`).
- **`IsAlreadyProcessed(event string, id string) bool`**: il metodo deve permettere di verificare se un evento è già stato elaborato. Prende come parametri il nome o tipo dell'evento (`event`) e il suo identificativo univoco (`id`). Restituisce `true` se l'evento è già stato processato, `false` altrimenti.

3.4.10.19 - IdempotentRepositoryImpl

Questa struttura implementa l'interfaccia **IIdempotentRepository**, vedi la Sezione 3.4.10.18.

Descrizione degli attributi della struttura:

- **`mutex sync.Mutex`**: mutex utilizzato per garantire la sicurezza dei dati in caso di accesso concorrente;
- **`m map[string]map[string]struct{}`**: mappa annidata che associa un evento (stringa) a un'altra mappa contenente gli identificativi univoci degli eventi già processati.

Descrizione dei metodi invocabili dalla struttura:

- **`NewIdempotentRepositoryImpl() *IdempotentRepositoryImpl`**: rappresenta il costruttore della struttura. Inizializza la mappa `m` come una mappa vuota;
- **`SaveEventID(event string, id string)`**: salva l'identificativo di un evento per garantire che non venga elaborato più volte. Se l'evento non esiste nella mappa, viene creato un nuovo record per esso;
- **`IsAlreadyProcessed(event string, id string) bool`**: verifica[G] se un evento è già stato elaborato. Restituisce `true` se l'evento è già stato processato, `false` altrimenti.

3.4.10.20 - IDempotentAdapter

Adapter che mette in comunicazione la *business logic* del microservizio **Warehouse**[G] con la *persistence logic* per la gestione dell'idempotenza.

Implementa l'interfaccia **IIdempotentPort**, vedi la Sezione 3.4.10.17.

Descrizione degli attributi della struttura:

- **`repo IIdempotentRepository`**: l'*Adapter* possiede un attributo alla struttura rappresentante la *persistence logic* per la gestione dell'idempotenza. Per le informazioni riguardo IIdempotentRepository vedere la Sezione 3.4.10.18.

Descrizione dei metodi invocabili dalla struttura:

- **`NewIDempotentAdapter(repo IIdempotentRepository) *IDempotentAdapter`**: costruttore dell'*Adapter*. Inizializza l'attributo `repo` con quello passato come parametro al costruttore;

- **`SaveEventID(cmd port.IdempotentCmd)`**: converte il comando per il salvataggio dell'identificativo di un evento in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata;
- **`IsAlreadyProcessed(cmd port.IdempotentCmd) bool`**: converte il comando per la verifica[G] se un evento è già stato elaborato in valori da fornire alla *persistence logic*, quindi richiama la *persistence logic* ad eseguire l'operazione desiderata e ritorna il risultato.

3.4.10.21 - StockPersistenceAdapter

Adapter che mette in comunicazione la *business logic* di Warehouse[G] con la *persistence logic* dello stesso.

Implementa le seguenti interfacce (porte):

- **IApplyStockUpdatePort**, Sezione 3.4.10.12;
- **IGetStockPort**, Sezione 3.4.10.13;
- **IApplyReservationEventPort**, Sezione 3.4.10.8;
- **IGetReservationPort**, Sezione 3.4.10.14.

Descrizione degli attributi della struttura:

- **`stockRepo IStockRepository`**: l'*Adapter* possiede un attributo alla struttura rappresentante la *persistence logic* di Warehouse[G]. Per le informazioni riguardo IStockRepository vedere la Sezione 3.4.10.2.

Descrizione dei metodi invocabili dalla struttura:

- **`NewStockPersistenceAdapter(stockRepo IStockRepository) *StockPersistenceAdapter`**: costruttore dell'*Adapter*. Inizializza l'attributo `stockRepo` con quello passato come parametro al costruttore;
- **`ApplyStockUpdate(goods []model.GoodStock) error`**: converte l'aggiornamento dello stock[G] in valori da fornire alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata;
- **`ApplyReservationEvent(reservation model.Reservation) error`**: converte l'evento di prenotazione in valori da fornire alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata. Ritorna un errore in caso di fallimento;
- **`ApplyOrderFilled(reservation model.Reservation) error`**: gestisce l'applicazione degli ordini con stato `Filled`, liberando le quantità riservate. Ritorna un errore in caso di fallimento;
- **`GetStock(goodId model.GoodID) model.GoodStock`**: converte la richiesta di ottenimento della quantità di una merce in valori da fornire alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata e ritorna un oggetto `GoodStock`;
- **`GetFreeStock(goodId model.GoodID) model.GoodStock`**: converte la richiesta di ottenimento della quantità libera di una merce in valori da fornire alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata e ritorna un oggetto `GoodStock`;
- **`GetReservation(reservationId model.ReservationID) (model.Reservation, error)`**: converte la richiesta di ottenimento dei dettagli di una prenotazione in valori da fornire

alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata. Ritorna un oggetto `Reservation` e un eventuale errore.

3.4.10.22 - IApplyCatalogUpdatePort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di applicare un aggiornamento del catalogo.

Descrizione dei metodi dell'interfaccia:

- **`ApplyCatalogUpdate(GoodInfo)`**: il metodo deve permettere di applicare un aggiornamento del catalogo.

3.4.10.23 - IGetGoodPort

Rappresenta la porta che consente alla *business logic* di comunicare alla *persistence logic* la volontà di ottenere le informazioni di una merce.

Descrizione dei metodi dell'interfaccia:

- **`GetGood(GoodId) GoodInfo`**: il metodo deve permettere di ottenere le informazioni di una merce tramite il suo ID.

3.4.10.24 - CatalogPersistenceAdapter

Adapter che mette in comunicazione la *business logic* di Warehouse[G] con la *persistence logic* dello stesso.

Implementa le seguenti interfacce (porte):

- **IApplyCatalogUpdatePort**, Sezione 3.4.10.22;
- **IGetGoodPort**, Sezione 3.4.10.23.

Descrizione degli attributi della struttura:

- **`catalogRepo ICatalogRepository`**: l'*Adapter* possiede un attributo alla struttura rappresentante la *persistence logic* di Warehouse[G]. Per le informazioni riguardo ICatalogRepository vedere la Sezione 3.4.10.4.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogPersistenceAdapter(catalogRepo ICatalogRepository) *CatalogPersistenceAdapter`**: costruttore dell'*Adapter*. Inizializza l'attributo `catalogRepo` con quello passato come parametro al costruttore;
- **`ApplyCatalogUpdate(good model.GoodInfo) error`**: converte l'aggiornamento del catalogo in valori da fornire alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata;
- **`GetGood(goodId model.GoodID) GoodInfo`**: converte la richiesta di ottenimento delle informazioni di una merce in valori da fornire alla *persistence Logic*, quindi richiama la *persistence Logic* ad eseguire l'operazione desiderata.

3.4.10.25 - IRemoveStockUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di rimuovere una quantità di una merce dal magazzino.

Descrizione dei metodi dell'interfaccia:

- **`RemoveStock(context.Context, RemoveStockCmd) error`**: il metodo deve permettere di rimuovere una quantità di una merce dal magazzino.

3.4.10.26 - IAddStockUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di aggiungere una quantità di una merce al magazzino.

Descrizione dei metodi dell'interfaccia:

- **`AddStock(context.Context, AddStockCmd) error`**: il metodo deve permettere di aggiungere una quantità di una merce al magazzino.

3.4.10.27 - ManageStockService

Si occupa di gestire la *business logic* per l'aggiunta e la rimozione di stock[G] nel microservizio *Warehouse*[G].

Implementa le seguenti interfacce (*Use Case*):

- **IAddStockUseCase**, Sezione 3.4.10.26;
- **IRemoveStockUseCase**, Sezione 3.4.10.25.

Descrizione degli attributi della struttura:

- **`createStockUpdatePort port.ICreateStockUpdatePort`**: vedere la descrizione alla Sezione 3.4.10.6;
- **`getGoodPort port.IGetGoodPort`**: vedere la descrizione alla Sezione 3.4.10.23;
- **`getStockPort port.IGetStockPort`**: vedere la descrizione alla Sezione 3.4.10.13;
- **`transactionPort port.ITransactionPort`**: vedere la descrizione alla Sezione 3.4.10.15.

Descrizione dei metodi invocabili dalla struttura:

- **`NewManageStockService(p ManageStockServiceParams) *ManageStockService`**: Costruttore della struttura. Le porte devono essere fornite mediante la struttura `ManageStockServiceParams`, dotata di `fx.In`;
- **`RemoveStock(ctx context.Context, cmd port.RemoveStockCmd) error`**: prende un *Command* per la richiesta di rimozione di stock[G] e utilizza la porta adibita allo scopo per svolgere la richiesta. Ritorna quindi l'esito dell'operazione;
- **`AddStock(ctx context.Context, cmd port.AddStockCmd) error`**: prende un *Command* per la richiesta di aggiunta di stock[G] e utilizza la porta adibita allo scopo per svolgere la richiesta. Ritorna quindi l'esito dell'operazione.

3.4.10.28 - StockController

Lo **StockController** gestisce l'*application logic* per le operazioni di aggiunta e rimozione di stock[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`addStockUseCase port.IAddStockUseCase`**: rappresenta il caso d'uso[G] per l'aggiunta di stock[G] vedere la Sezione 3.4.10.26;
- **`removeStockUseCase port.IRemoveStockUseCase`**: rappresenta il caso d'uso[G] per la rimozione di stock[G] vedere la Sezione 3.4.10.25.

Descrizione dei metodi invocabili dalla struttura:

- **`NewStockController(addStockUseCase port.IAddStockUseCase, removeStockUseCase port.IRemoveStockUseCase, mp MetricParams) *StockController`**: costruttore della struttura. Inizializza gli attributi `addStockUseCase` e `removeStockUseCase` con i valori passati come parametri e la telemetria con quanto contenuto in `MetricParams`;
- **`RemoveStockHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce i messaggi per la rimozione di stock[G]. Ritorna un errore in caso l'operazione non venga completata correttamente;
- **`AddStockHandler(ctx context.Context, msg *nats.Msg) error`**: gestisce i messaggi per l'aggiunta di stock[G]. Ritorna un errore in caso l'operazione non venga completata correttamente.

3.4.10.29 - IApplyStockUpdateUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di applicare un aggiornamento dello *stock*[G].

Descrizione dei metodi dell'interfaccia:

- **`ApplyStockUpdate(StockUpdateCmd)`**: il metodo deve permettere di applicare un aggiornamento dello stock[G]. Prende come parametro un oggetto `StockUpdateCmd` che contiene tutte le informazioni necessarie per l'aggiornamento dello stock[G].

3.4.10.30 - ApplyStockUpdateService

Si occupa di gestire la *business logic* per l'applicazione degli aggiornamenti dello stock[G] nel microservizio *Warehouse*[G].

Implementa le seguenti interfacce (*Use Case*):

- **IApplyStockUpdateUseCase**, Sezione 3.4.10.29.

Descrizione degli attributi della struttura:

- **`applyStockUpdatePort port.IApplyStockUpdatePort`**: vedere la descrizione alla Sezione 3.4.10.12;
- **`idempotentPort port.IIdempotentPort`**: vedere la descrizione alla Sezione 3.4.10.17.
- **`transactionPort port.ITransactionPort`**: vedere la descrizione alla Sezione 3.4.10.15.

Descrizione dei metodi invocabili dalla struttura:

- **`NewApplyStockUpdateService(applyStockUpdatePort port.IApplyStockUpdatePort, idempotentPort port.IIdempotentPort, transactionPort port.ITransactionPort)`**

***ApplyStockUpdateService**: Costruttore della struttura. Le porte utilizzate vengono fornite come parametro al costruttore;

- **ApplyStockUpdate(cmd port.StockUpdateCmd)**: prende un *Command* per la richiesta di applicazione di un aggiornamento dello stock[G] e utilizza la porta adibita allo scopo per svolgere la richiesta.

3.4.10.31 - StockUpdateListener

Lo **StockUpdateListener** gestisce l'*application logic* per l'ascolto degli aggiornamenti dello stock[G] nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **applyStockUpdateUseCase port.IApplyStockUpdateUseCase**: rappresenta il caso d'uso[G] per l'applicazione degli aggiornamenti dello stock[G] vedere la Sezione 3.4.10.29.

Descrizione dei metodi invocabili dalla struttura:

- **NewStockUpdateListener(applyStockUpdateUseCase port.IApplyStockUpdateUseCase, mp MetricParams) *StockUpdateListener**: costruttore della struttura. Inizializza l'attributo `applyStockUpdateUseCase` con il valore passato come parametro e la telemetria con quanto contenuto in `MetricParams`;
- **ListenStockUpdate(ctx context.Context, msg jetstream.Msg) error**: gestisce i messaggi per l'applicazione degli aggiornamenti dello stock[G]. Ritorna un errore in caso l'operazione non venga completata correttamente.

3.4.10.32 - IApplyCatalogUpdateUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di applicare un aggiornamento del catalogo.

Descrizione dei metodi dell'interfaccia:

- **ApplyCatalogUpdate(CatalogUpdateCmd)**: il metodo deve permettere di applicare un aggiornamento del catalogo.

3.4.10.33 - ApplyCatalogUpdateService

Si occupa di gestire la *business logic* per l'applicazione degli aggiornamenti del catalogo nel microservizio *Warehouse*[G].

Implementa le seguenti interfacce (*Use Case*):

- **IApplyCatalogUpdateUseCase**, Sezione 3.4.10.32.

Descrizione degli attributi della struttura:

- **applyCatalogUpdatePort port.IApplyCatalogUpdatePort**: vedere la descrizione alla Sezione 3.4.10.22.

Descrizione dei metodi invocabili dalla struttura:

- **NewApplyCatalogUpdateService(applyCatalogUpdatePort port.IApplyCatalogUpdatePort, transactionPort port.ITransactionPort) *ApplyCatalogUpdateService**: Costruttore della struttura. La porte deveno essere fornite come parametro al costruttore;

- **`ApplyCatalogUpdate(cmd port.CatalogUpdateCmd)`**: prende un *Command* per la richiesta di applicazione di un aggiornamento del catalogo e utilizza la porta adibita allo scopo per svolgere la richiesta.

3.4.10.34 - IConfirmOrderUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di confermare un ordine[G].

Descrizione dei metodi dell'interfaccia:

- **`ConfirmOrder(context.Context, ConfirmOrderCmd) error`**: il metodo deve permettere di confermare un ordine[G], prendendo come parametri il contesto e un oggetto di tipo `ConfirmOrderCmd`. Deve restituire un errore in caso di fallimento.

3.4.10.35 - IConfirmTransferUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di confermare un trasferimento[G].

Descrizione dei metodi dell'interfaccia:

- **`ConfirmTransfer(context.Context, ConfirmTransferCmd) error`**: il metodo deve permettere di confermare un trasferimento[G], prendendo come parametri il contesto e un oggetto di tipo `ConfirmTransferCmd`. Deve restituire un errore in caso di fallimento.

3.4.10.36 - ICreateReservationUseCase

Rappresenta l'interfaccia che permette all'*application logic* di comunicare alla *business logic* la volontà di creare una prenotazione.

Descrizione dei metodi dell'interfaccia:

- **`CreateReservation(context.Context, CreateReservationCmd) (CreateReservationResponse, error)`**: il metodo deve permettere di creare una prenotazione, prendendo come parametri il contesto e un oggetto di tipo `CreateReservationCmd`. Deve restituire un oggetto di tipo `CreateReservationResponse` contenente l'identificativo della prenotazione creata e un eventuale errore in caso di fallimento.

3.4.10.37 - ManageReservationService

Si occupa di gestire la *business logic* per la gestione delle prenotazioni nel microservizio *Warehouse*[G].

Implementa le seguenti interfacce (*Use Case*):

- **ICreateReservationUseCase**, Sezione 3.4.10.36;
- **IApplyReservationUseCase**, Sezione 3.4.10.9;
- **IConfirmOrderUseCase**, Sezione 3.4.10.34;
- **IConfirmTransferUseCase**, Sezione 3.4.10.35.

Descrizione degli attributi della struttura:

- **`createReservationEventPort port.IStoreReservationEventPort`**: rappresenta la porta per memorizzare gli eventi di creazione delle prenotazioni; vedere la Sezione 3.4.10.7;
- **`applyReservationEventPort port.IApplyReservationEventPort`**: rappresenta la porta per applicare gli eventi di prenotazione; vedere la Sezione 3.4.10.8;

- **`getReservationPort port.IGetReservationPort`**: rappresenta la porta per ottenere i dettagli di una prenotazione; vedere la Sezione 3.4.10.14;
- **`getStockPort port.IGetStockPort`**: rappresenta la porta per ottenere la quantità di una merce presente nel magazzino; vedere la Sezione 3.4.10.13;
- **`createStockUpdatePort port.ICreateStockUpdatePort`**: rappresenta la porta per creare aggiornamenti dello stock[G] vedere la Sezione 3.4.10.6;
- **`idempotentPort port.IIdempotentPort`**: rappresenta la porta per gestire operazioni idempotenti; vedere la Sezione 3.4.10.17;
- **`cfg *config.WarehouseConfig`**: rappresenta la configurazione del microservizio *Warehouse*[G].
- **`transactionPort port.ITransactionPort`**: rappresenta la porta per gestire le transazioni; vedere la Sezione 3.4.10.15;

Descrizione dei metodi invocabili dalla struttura:

- **`NewManageReservationService(p ManageReservationServiceParams) *ManageReservationService`**: Costruttore della struttura. Gli attributi della struttura vengono inizializzati utilizzando la struttura **`ManageReservationServiceParams`**, che utilizza l'istruzione `fx.in` per permettere al *framework* **fx** di fornire automaticamente le dipendenze necessarie;
- **`CreateReservation(ctx context.Context, cmd port.CreateReservationCmd) (CreateReservationResponse, error)`**: prende un comando per la creazione di una prenotazione e utilizza le porte adibite per svolgere la richiesta. Ritorna la risposta alla creazione della prenotazione o un errore in caso di fallimento;
- **`ApplyReservationEvent(cmd port.ApplyReservationEventCmd) error`**: prende un comando per applicare un evento di prenotazione e utilizza le porte adibite per svolgere la richiesta. Ritorna un errore in caso di fallimento;
- **`ConfirmOrder(ctx context.Context, cmd port.ConfirmOrderCmd) error`**: prende un comando per confermare un ordine[G] e utilizza le porte adibite per svolgere la richiesta. Ritorna un errore in caso di fallimento;
- **`ConfirmTransfer(ctx context.Context, cmd port.ConfirmTransferCmd) error`**: prende un comando per confermare un trasferimento[G] e utilizza le porte adibite per svolgere la richiesta. Ritorna un errore in caso di fallimento.

3.4.10.38 - CatalogListener

Il **CatalogListener** gestisce l'*Application Logic* per l'ascolto degli aggiornamenti del catalogo nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`applyCatalogUpdateUseCase port.IApplyCatalogUpdateUseCase`**: rappresenta il caso d'uso[G] per l'applicazione degli aggiornamenti del catalogo; vedere la Sezione 3.4.10.32.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogListener(applyCatalogUpdateUseCase port.IApplyCatalogUpdateUseCase, mp MetricParams) *CatalogListener`**: costruttore della struttura. Inizializza l'attributo `applyCatalogUpdateUseCase` con il valore passato come parametro e la telemetria con quanto contenuto in `MetricParams`;

- **`ListenGoodUpdate(ctx context.Context, msg jetstream.Msg) error`**: gestisce i messaggi per l'applicazione degli aggiornamenti del catalogo. Ritorna un errore in caso l'operazione non venga completata correttamente.

3.4.10.39 - HealthCheckController

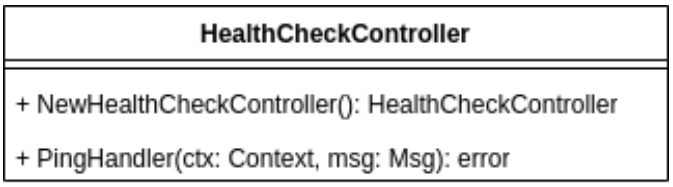

Figura 143: Warehouse - HealthCheckController

Il **HealthCheckController** gestisce l'*Application Logic* per le operazioni di controllo dello stato di salute del microservizio **Warehouse**[G].

Descrizione dei metodi invocabili dalla struttura:

- **`NewHealthCheckController() *HealthCheckController`**: costruttore della struttura. Inizializza gli attributi necessari per il controllo dello stato di salute;
- **`PingHandler(ctx context.Context, msg nats.Msg) error`**: gestisce le richieste di controllo dello stato di salute del microservizio. Ritorna un errore in caso l'operazione non venga completata correttamente.

3.4.10.40 - ReservationController

Il **ReservationController** gestisce l'*Application Logic* per le operazioni di creazione delle prenotazioni nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`createReservationUseCase port.ICreateReservationUseCase`**: rappresenta il caso d'uso[G] per la creazione delle prenotazioni; vedere la Sezione 3.4.10.36.

Descrizione dei metodi invocabili dalla struttura:

- **`NewReservationController(createReservationUseCase ICreateReservationUseCase, mp MetricParams) *ReservationController`**: costruttore della struttura. Inizializza l'attributo `createReservationUseCase` con il valore passato come parametro e la telemetria con quanto contenuto in `MetricParams`;
- **`CreateReservationHandler(ctx context.Context, msg nats.Msg) error`**: gestisce le richieste di creazione delle prenotazioni, trasformando i dati ricevuti in un comando e delegando l'operazione al caso d'uso[G]. Risponde con un messaggio di successo o errore.

3.4.10.41 - ReservationEventListener

Il **ReservationEventListener** gestisce l'*Application Logic* per l'ascolto degli eventi di prenotazione nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`applyReservationEventUseCase port.IApplyReservationUseCase`**: rappresenta il caso d'uso[G] per l'applicazione degli eventi di prenotazione; vedere la Sezione 3.4.10.9.

Descrizione dei metodi invocabili dalla struttura:

- **`NewReservationEventListener(applyReservationEventUseCase IApplyReservationUseCase, mp MetricParams) *ReservationEventListener`**: costruttore della struttura. Inizializza l'attributo `applyReservationEventUseCase` con il valore passato come parametro e la telemetria con quanto contenuto in `MetricParams`;
- **`ListenReservationEvent(ctx context.Context, msg jetstream.Msg) error`**: gestisce i messaggi per l'applicazione degli eventi di prenotazione. Decodifica il messaggio ricevuto in un oggetto `ReservationEvent`, lo trasforma in un comando `ApplyReservationEventCmd` e delega l'operazione al caso d'uso[G]. Ritorna un errore in caso l'operazione non venga completata correttamente.

3.4.10.42 - OrderUpdateListener

Il **OrderUpdateListener** gestisce l'*Application Logic* per l'ascolto degli aggiornamenti degli ordini e dei trasferimenti nel microservizio **Warehouse**[G].

Descrizione degli attributi della struttura:

- **`confirmOrderUseCase port.IConfirmOrderUseCase`**: rappresenta il caso d'uso[G] per la conferma degli ordini; vedere la Sezione 3.4.10.34;
- **`confirmTransferUseCase port.IConfirmTransferUseCase`**: rappresenta il caso d'uso[G] per la conferma dei trasferimenti; vedere la Sezione 3.4.10.35.

Descrizione dei metodi invocabili dalla struttura:

- **`NewOrderUpdateListener(confirmOrderUseCase port.IConfirmOrderUseCase, confirmTransferUseCase port.IConfirmTransferUseCase) *OrderUpdateListener`**: costruttore della struttura. Inizializza gli attributi `confirmOrderUseCase` e `confirmTransferUseCase` con i valori passati come parametri;
- **`ListenOrderUpdate(ctx context.Context, msg jetstream.Msg) error`**: gestisce i messaggi per l'aggiornamento degli ordini. Decodifica il messaggio ricevuto in un oggetto `OrderUpdate`, lo trasforma in un comando `ConfirmOrderCmd` e delega l'operazione al caso d'uso[G] `confirmOrderUseCase`. Ritorna un errore in caso l'operazione non venga completata correttamente;
- **`ListenTransferUpdate(ctx context.Context, msg jetstream.Msg) error`**: gestisce i messaggi per l'aggiornamento dei trasferimenti. Decodifica il messaggio ricevuto in un oggetto `TransferUpdate`, lo trasforma in un comando `ConfirmTransferCmd` e delega l'operazione al caso d'uso[G] `confirmTransferUseCase`. Ritorna un errore in caso l'operazione non venga completata correttamente.

3.4.11 - Notification

Il microservizio **Notification** si occupa di controllare periodicamente la quantità di merce presente nel sistema, confrontandola con una quantità di riferimento impostata dall'utente. Se la quantità configurata dell'utente non è soddisfatta, allora viene inviata una notifica in uno stream dedicato, seguita da un secondo messaggio quando la quantità ritorna nei livelli nominali.

3.4.11.1 - Oggetti comuni del microservizio

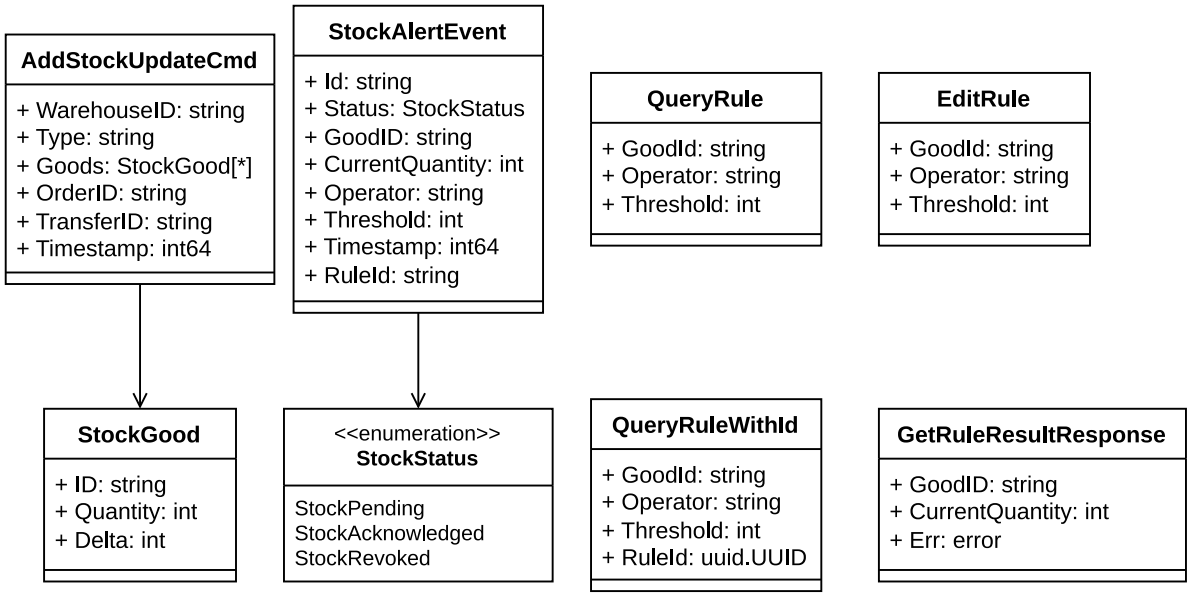

Figura 144: Tipi comuni a tutto il microservizio Notification

In Go, gli *import* devono obbligatoriamente formare un grafo aciclico; di conseguenza questi tipi, che vengono utilizzati sia dalla *business logic* che dagli *adapter*, sono stati estratti in un *package* separato per evitare errori di compilazione.

3.4.11.1.1 - AddStockUpdateCmd

Questo tipo viene inviato alla *business logic* quando arriva uno *stock update* da registrare.

Descrizione degli attributi della struttura:

- **`WarehouseID string`** - Id del magazzino da cui proviene lo *stock update*;
- **`Type string`** - Tipo dello stock update. Valori possibili: `add`, `remove`;
- **`Goods []StockGood`** - Lista di aggiornamenti effettuati;
- **`OrderID string`** - Id dell'ordine;
- **`TransferID string`** - Id del trasferimento;
- **`Timestamp int64`** - Istante temporale in cui lo stock update è avvenuto, come Unix Timestamp.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.1.2 - StockGood

Una singola merce all'interno di un AddStockUpdateCmd.

Descrizione degli attributi della struttura:

- **`ID string`** - Id della merce;

- **`Quantity int`** - Nuova quantità presente **per il magazzino specificato**;
- **`Delta int`** - Cambiamento nella quantità immagazzinata. Negativo se è diminuita.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.1.3 - StockAlertEvent

Questo tipo rappresenta una notifica che sta per essere inviata.

Descrizione degli attributi della struttura:

- **`Id string`** - Id univoco della notifica;
- **`Status: StockStatus`** - Stato della notifica. Valori possibili: `Pending`, `Acknowledged`, e `Revoked`;
- **`GoodID string`** - Id della merce a cui la notifica fa riferimento;
- **`CurrentQuantity int`** - Quantità presente in magazzino al momento dell'invio della notifica;
- **`Operator string`** - Operatore della notifica;
- **`Threshold int`** - Quantità limite contro il quale è stata confrontata la quantità immagazzinata;
- **`Timestamp int64`** - Istante temporale in cui il messaggio è stato inviato, come Unix Timestamp;
- **`RuleId string`** - Id della regola che ha scatenato questa notifica.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.1.4 - StockStatus

Questo tipo è una *type definition* che punta al tipo primitivo `string`. Nonostante ciò, i due tipi sono diversi, ed è possibile convertire da uno all'altro solo inserndo un *cast* esplicito.

3.4.11.1.5 - QueryRule

Questo tipo è utilizzato all'interno della *business logic* per descrivere le regole utilizzate per decidere quando mandare una notifica.

Descrizione degli attributi della struttura:

- **`GoodId string`** - L'id del Good (merce) che si vuole controllare;
- **`Operator string`** - Operatore da usare. Valori possibili: <, >, <=, >=;
- **`Threshold int`** - Quantità limite contro il quale verrà confrontata la quantità immagazzinata.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.1.6 - QueryRuleWithId

Questo tipo è identico a QueryRule, se non per l'aggiunta del campo `RuleId`.

Descrizione degli attributi della struttura:

- **`GoodId string`** - L'id del Good (merce) che si vuole controllare;
- **`Operator string`** - Operatore da usare. Valori possibili: <, >, <=, >=;

- **`Threshold int`** - Quantità limite contro il quale verrà confrontata la quantità immagazzinata;
- **`RuleId: uuid.UUID`** - Id della regola.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.1.7 - EditRule

Questo tipo è identico a QueryRule, tranne per il fatto che tutti i suoi campi sono opzionali (notare gli asterischi, che indicano puntatori, che in Go possono anche essere nulli). L'intenzione è di rendere possibile evitare la modifica dei campi che l'utente non desidera ritoccare.

Descrizione degli attributi della struttura:

- **`GoodId *string`** - L'id del Good (merce) che si vuole controllare;
- **`Operator *string`** - Operatore da usare. Valori possibili: <, >, <=, >=;
- **`Threshold *int`** - Quantità limite contro il quale verrà confrontata la quantità immagazzinata.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.1.8 - GetRuleResultResponse

Questo tipo è utilizzato come risultato dell'operazione `GetCurrentQuantityByGoodID()`, proveniente dall'interfaccia

Descrizione degli attributi della struttura:

- **`GoodID string`** - Identificatore della merce;
- **`CurrentQuantity int`** - Quantità attuale della merce richiesta;
- **`Err: error`** - Eventuale errore incontrato durante l'operazione.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.2 - *Business logic*

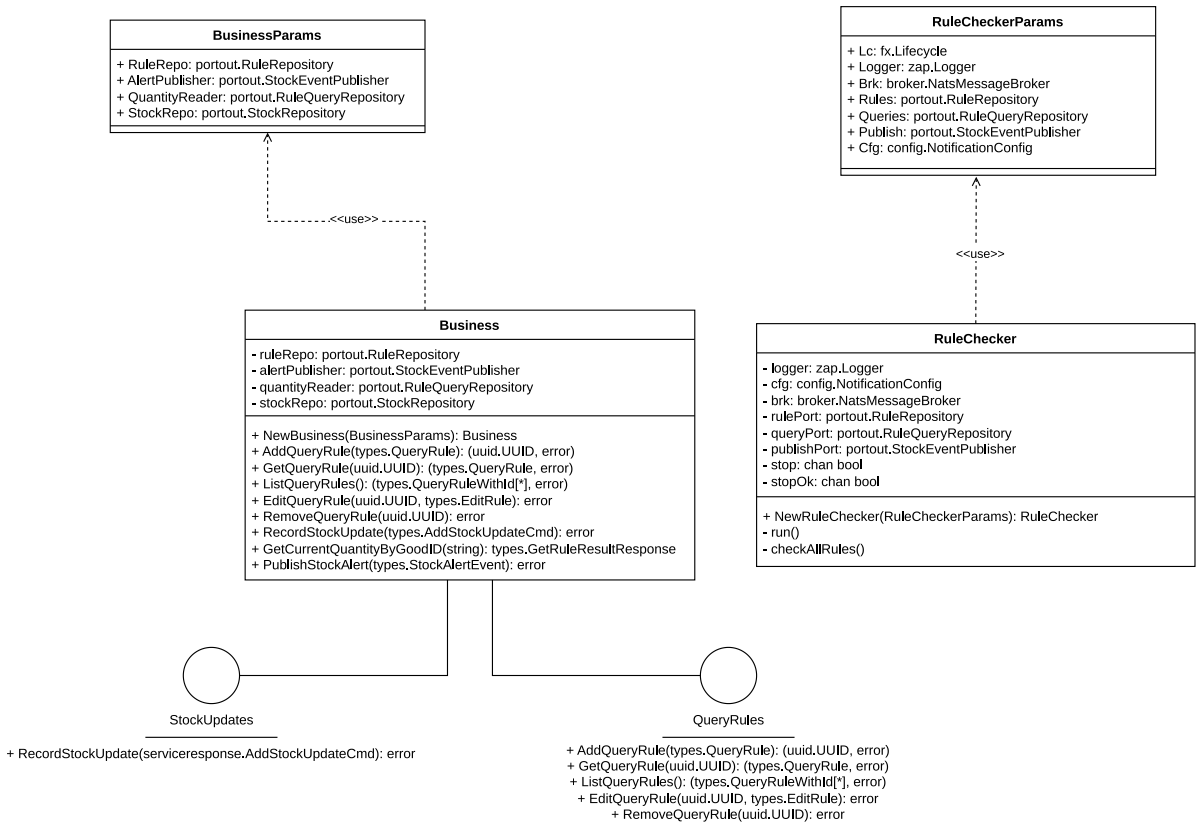

Figura 145: *Business logic* del microservizio Notification

3.4.11.2.1 - BusinessParams

Parametri necessari per costruire un'istanza di Business

Descrizione degli attributi della struttura:

- **`RuleRepo portout.RuleRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`AlertPublisher portout.StockEventPublisher`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`QuantityReader portout.RuleQueryRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`StockRepo portout.StockRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.2.2 - business.Business

Business logic del microservizio di notifica.

Descrizione degli attributi della struttura:

- **`ruleRepo portout.RuleRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`alertPublisher portout.StockEventPublisher`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;

- **`quantityReader portout.RuleQueryRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`stockRepo portout.StockRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewBusiness(BusinessParams) *Business`** - Costruttore della struttura;
- **`AddQueryRule(types.QueryRule) (uuid.UUID, error)`** - Aggiunge una regola di notifica;
- **`GetQueryRule(uuid.UUID) (types.QueryRule, error)`** - Recupera una regola di notifica dato il suo identificatore;
- **`ListQueryRules() ([]types.QueryRuleWithId, error)`** - Recupera tutte le regole di notifica;
- **`EditQueryRule(uuid.UUID, types.EditRule) error`** - Modifica una regola di notifica;
- **`RemoveQueryRule(uuid.UUID) error`** - Rimuove una regola di notifica;
- **`RecordStockUpdate(*types.AddStockUpdateCmd) error`** - Salva uno *stock_update* in maniera permanente;
- **`GetCurrentQuantityByGoodID(string) *types.GetRuleResultResponse`** - Recupera la quantità attuale di una merce, dato il suo identificatore;
- **`PublishStockAlert(types.StockAlertEvent) error`** - Pubblica una notifica.

3.4.11.2.3 - RuleCheckerParams

Parametri necessari per costruire un'istanza di RuleChecker

Descrizione degli attributi della struttura:

- **`Lc fx.Lifecycle`** - Gestisce il ciclo di vita dei componenti dell'applicazione;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log;
- **`Brk *broker.NatsMessageBroker`** - Astrazione che incapsula la connessione a NATS;
- **`Rules portout.RuleRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`Queries portout.RuleQueryRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`Publish portout.StockEventPublisher`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`Cfg *config.NotificationConfig`** - Configurazione del microservizio.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.2.4 - business.RuleChecker

Componente che implementa un controllo periodico delle regole di notifica, e si occupa di mandare le regole in caso serva.

Descrizione degli attributi della struttura:

- **`logger *zap.Logger`** - Necessario per raccogliere i log;
- **`cfg *config.NotificationConfig`** - Configurazione del microservizio;
- **`brk *broker.NatsMessageBroker`** - Astrazione che incapsula la connessione a NATS;
- **`rulePort portout.RuleRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;

- **`queryPort portout.RuleQueryRepository`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`publishPort portout.StockEventPublisher`** - *Port-out* che verrà utilizzata dalla *business logic* per parlare con gli *adapter-out*;
- **`stop chan bool`** - Canale usato per ricevere un messaggio quando l'applicazione sta venendo terminata;
- **`stopOk chan bool`** - Canale su cui viene inviata una conferma quando *RuleChecker* sta venendo terminato.

Descrizione dei metodi invocabili dalla struttura:

- **`NewRuleChecker(RuleCheckerParams) *RuleChecker`** - Costruttore della struttura;
- **`run()`** - Blocca la *goroutine* corrente ed avvia il controllo delle regole periodico;
- **`checkAllRules()`** - Controlla le regole. Viene chiamata da `run()`.

3.4.11.3 - *Adapter-in* e *Port-in*

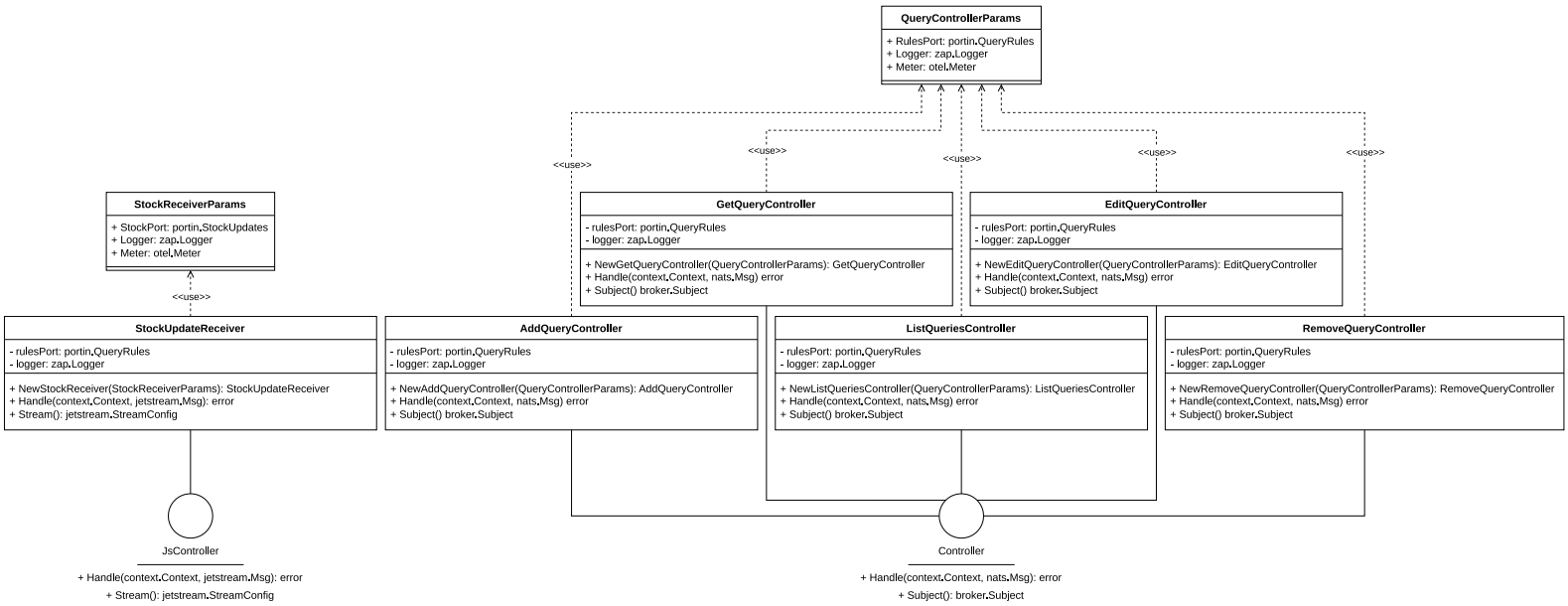

Figura 146: *Adapter-in* di Notification

Questi tipi sono stati utilizzati per realizzare la parte di «ingresso» dell'architettura esagonale. Nella sezione corrente sono inclusi i tipi dei *package* `adapterin` e `portin`.

3.4.11.3.1 - StockReceiverParams

Parametri in ingresso necessari per costruire un'istanza di StockUpdateReceiver.

Descrizione degli attributi della struttura:

- **`StockPort portin.StockUpdates`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log;
- **`Meter metric.Meter`** - Necessario per raccogliere le metriche.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.3.2 - StockUpdateReceiver

Questo *Adapter-in* è incaricato di ricevere da *NATS* gli *stock update*. Implementa l'interfaccia JsController.

Descrizione degli attributi della struttura:

- **`stockPort portin.StockUpdates`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`Handle(context.Context, jetstream.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Stream() jetstream.StreamConfig`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.3 - JsController

Questa interfaccia accomuna tutti i *controller* che si interfacciano con *Jetstream*, in contrapposizione a quelli che si interfacciano con *NATS Core*, rappresentati da Controller.

Descrizione dei metodi dall'interfaccia:

- **`Handle(context.Context, jetstream.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Stream() jetstream.StreamConfig`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.4 - AddQueryController

Questo controller gestisce l'aggiunta di regole al microservizio. Implementa l'interfaccia Controller.

Descrizione degli attributi della struttura:

- **`rulesPort portin.QueryRules`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAddQueryController(QueryControllersParams) *AddQueryController`** - Costruttore della struttura;
- **`Handle(context.Context, *nats.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Subject() broker.Subject`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.5 - GetQueryController

Questo controller gestisce la modifica di regole del microservizio. Implementa l'interfaccia Controller.

Descrizione degli attributi della struttura:

- **`rulesPort portin.QueryRules`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;

- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetQueryController(QueryControllersParams) *GetQueryController`** - Costruttore del tipla struttura;
- **`Handle(context.Context, *nats.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Subject() broker.Subject`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.6 - ListQueriesController

Questo controller gestisce la modifica di regole del microservizio. Implementa l'interfaccia Controller.

Descrizione degli attributi della struttura:

- **`rulesPort portin.QueryRules`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewListQueriesController(QueryControllersParams) *ListQueriesController`** - Costruttore del tipla struttura;
- **`Handle(context.Context, *nats.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Subject() broker.Subject`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.7 - EditQueryController

Questo controller gestisce la modifica di regole del microservizio. Implementa l'interfaccia Controller.

Descrizione degli attributi della struttura:

- **`rulesPort portin.QueryRules`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewEditQueryController(QueryControllersParams) *EditQueryController`** - Costruttore del tipla struttura;
- **`Handle(context.Context, *nats.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Subject() broker.Subject`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.8 - RemoveQueryController

Questo controller gestisce la modifica di regole del microservizio. Implementa l'interfaccia Controller.

Descrizione degli attributi della struttura:

- **`rulesPort portin.QueryRules`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewRemoveQueryController(QueryControllersParams) *RemoveQueryController`** - Costruttore del tipla struttura;
- **`Handle(context.Context, *nats.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Subject() broker.Subject`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.9 - Controller

Questa interfaccia accomuna tutti i *controller* che si interfacciano con *NATS Core*, in contrapposizione a quelli che si interfacciano con *Jetstream*, rappresentati da JsController.

Descrizione dei metodi dall'interfaccia:

- **`Handle(context.Context, *nats.Msg) error`** - Questo metodo viene chiamato quando l'*adapter* riceve un messaggio;
- **`Subject() broker.Subject`** - Questo metodo viene chiamato in fase di avvio del software, per capire quali messaggi esso debba gestire.

3.4.11.3.10 - QueryControllersParams

Parametri in ingresso necessari per costruire un'istanza di AddQueryController, GetQueryController, ListQueriesController, EditQueryController, o RemoveQueryController.

Descrizione degli attributi della struttura:

- **`RulesPort portin.QueryRules`** - La *Port-in* che verrà utilizzata per parlare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log;
- **`Meter metric.Meter`** - Necessario per raccogliere le metriche.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.4 - *Adapter-out* e *Port-out*

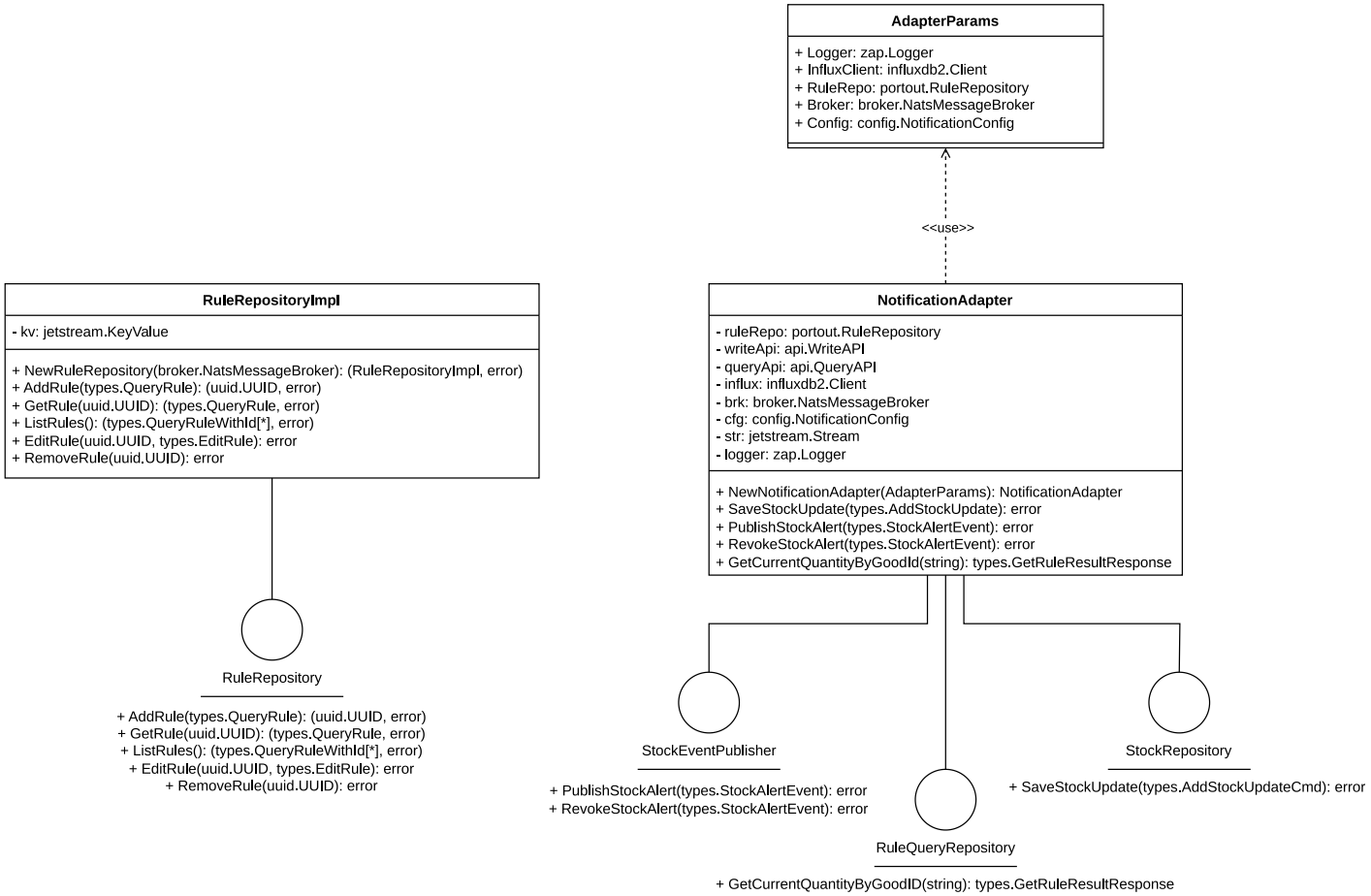

Figura 147: *Adapter-out* di Notification

Questi tipi sono stati utilizzati per realizzare la parte di «uscita» dell'architettura esagonale. Nella sezione corrente sono inclusi i tipi dei *package* `adapterout` e `portout`.

3.4.11.4.1 - AdapterParams

Parametri in ingresso necessari per costruire un'istanza di NotificationAdapter.

Descrizione degli attributi della struttura:

- **`Logger *zap.Logger`** - Necessario per raccogliere i log;
- **`InfluxClient: influxdb2.Client`** - *Client* per la connessione a *InfluxDB*;
- **`RuleRepo portout.RuleRepository`** - La *Port-out* che verrà utilizzata dalla *business logic* per parlare con questo componente;
- **`Broker *broker.NatsMessageBroker`** - Astrazione che contiene le connessioni a *NATS Core* e *Jetstream*, oltre a logica per gestire l'iscrizione a *subject* o *stream*;
- **`Config *config.NotificationConfig`** - Configurazione del microservizio.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.11.4.2 - NotificationAdapter

Adapter-out che realizza buona parte delle comunicazioni verso l'esterno effettuate dal microservizio. Implementa le interfacce RuleQueryRepository, StockEventPublisher e StockRepository.

Descrizione degli attributi della struttura:

- **`ruleRepo portout.RuleRepository`** - La *Port-out* che verrà utilizzata dalla *business logic* per parlare con questo componente;
- **`writeApi: api.WriteAPI`** - Connessione a *InfluxDB* dal lato «scrittura»;
- **`queryApi: api.QueryAPI`** - Connessione a *InfluxDB* dal lato «lettura»;
- **`influx: influxdb2.Client`** - *Client* per la connessione a *InfluxDB*;
- **`brk *broker.NatsMessageBroker`** - Astrazione che contiene le connessioni a *NATS Core* e *Jetstream*, oltre a logica per gestire l'iscrizione a *subject* o *stream*;
- **`cfg *config.NotificationConfig`** - Configurazione del microservizio;
- **`str: jetstream.Stream`** - *Stream* su cui questo adapter opera;
- **`logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewNotificationAdapter(AdapterParams) (*NotificationAdapter, error)`** - Costruttore della struttura;
- **`SaveStockUpdate(*types.AddStockUpdateCmd) error`** - Salva uno *stock update* all'interno di *InfluxDB*;
- **`PublishStockAlert(types.StockAlertEvent) error`** - Pubblica una notifica;
- **`RevokeStockAlert(types.StockAlertEvent) error`** - Segnala che una notifica è rientrata, ossia non è più presente nessuna condizione scatenante;
- **`GetCurrentQuantityByGoodID(string) *types.GetRuleResultResponse`** - Recupera la quantità presente in tutti i magazzini della merce con id specificato.

3.4.11.4.3 - RuleQueryRepository

Porta che permette di eseguire query su *InfluxDB*.

Descrizione dei metodi dall'interfaccia:

- **`GetCurrentQuantityByGoodID(string) *types.GetRuleResultResponse`** - Recupera la quantità presente in tutti i magazzini della merce con id specificato.

3.4.11.4.4 - StockEventPublisher

Porta che permette di pubblicare una notifica, o aggiornarne lo stato.

Descrizione dei metodi dall'interfaccia:

- **`PublishStockAlert(types.StockAlertEvent) error`** - Pubblica una notifica;
- **`RevokeStockAlert(types.StockAlertEvent) error`** - Segnala che una notifica è rientrata, ossia non è più presente nessuna condizione scatenante.

3.4.11.4.5 - StockRepository

Porta che permette di salvare *stock update* all'interno di *InfluxDB*.

Descrizione dei metodi dall'interfaccia:

- **`SaveStockUpdate(*types.AddStockUpdateCmd) error`** - Salva uno *stock update* all'interno di *InfluxDB*.

3.4.11.4.6 - RuleRepositoryImpl

Adapter-out che si occupa della persistenza delle regole di notifica. Implementa l'interfaccia RuleRepository.

Descrizione degli attributi della struttura:

- **`kv: jetstream.KeyValue`** - *Stream* su cui vengono salvate le regole.

Descrizione dei metodi invocabili dalla struttura:

- **`NewRuleRepository(*broker.NatsMessageBroker) (*RuleRepositoryImpl, error)`** - Costruttore della struttura;
- **`AddRule(types.QueryRule) (uuid.UUID, error)`** - Permette di aggiungere una regola;
- **`GetRule(uuid.UUID) (types.QueryRule, error)`** - Permette di ottenere una regola;
- **`ListRules() ([]types.QueryRuleWithId, error)`** - Permette di recuperare una lista delle regole;
- **`EditRule(uuid.UUID, types.EditRule) error`** - Permette di modificare una regola;
- **`RemoveRule(uuid.UUID) error`** - Permette di rimuovere una regola.

3.4.11.4.7 - RuleRepository

Porta che permette di persistere le regole di notifica.

Descrizione dei metodi dall'interfaccia:

- **`AddRule(types.QueryRule) (uuid.UUID, error)`** - Permette di aggiungere una regola;
- **`GetRule(uuid.UUID) (types.QueryRule, error)`** - Permette di ottenere una regola;
- **`ListRules() ([]types.QueryRuleWithId, error)`** - Permette di recuperare una lista delle regole;
- **`EditRule(uuid.UUID, types.EditRule) error`** - Permette di modificare una regola;
- **`RemoveRule(uuid.UUID) error`** - Permette di rimuovere una regola.

3.4.12 - API Gateway

Il microservizio **API Gateway** permette di interagire con il Sistema, esponendo un API HTTP e implementando autenticazione e autorizzazione.

3.4.12.1 - Oggetti comuni del microservizio

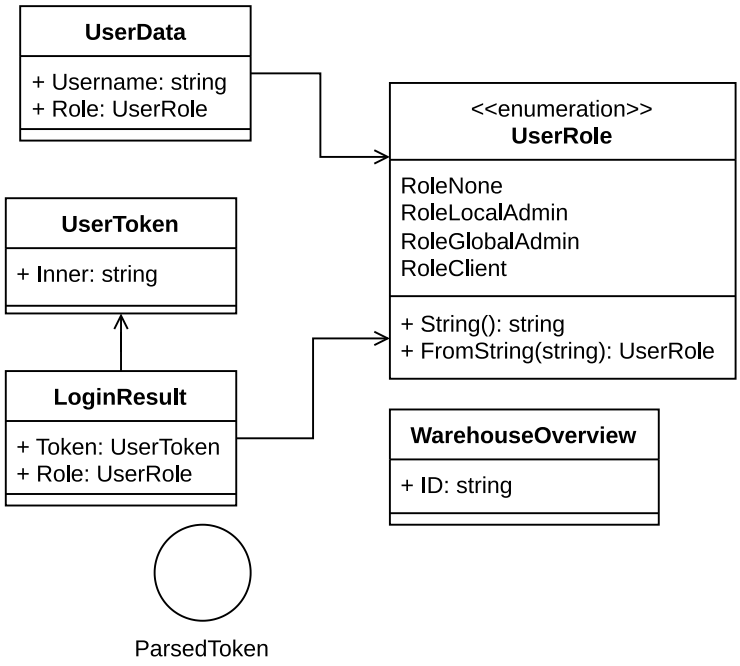

Figura 148: Tipi comuni a tutto il microservizio API Gateway

In Go, gli *import* devono obbligatoriamente formare un grafo aciclico; di conseguenza questi tipi, che vengono utilizzati sia dalla *business logic* che dagli *adapter*, sono stati estratti in un *package* separato per evitare errori di compilazione.

3.4.12.1.1 - types.UserRole

Questa enumerazione rappresenta il ruolo assegnato ad un utente.

Varianti dell'enumerazione:

- **`RoleNone`** - Nessun ruolo. Generalmente usato per indicare errori di autenticazione;
- **`RoleLocalAdmin`** - *admin* locale;
- **`RoleGlobalAdmin`** - *admin* globale;
- **`RoleClient`** - cliente.

Descrizione dei metodi invocabili dalla enumerazione:

- **`String() string`** - Ritorna la rappresentazione in stringa di questo ruolo;
- **`FromString(string) UserRole`** - Effettua il *parsing* della stringa fornita.

3.4.12.1.2 - types.UserToken

Questo tipo rappresenta un *token* di un utente autenticato.

Descrizione degli attributi della struttura:

- **`Inner string`** - Rappresentazione in stringa del *token*.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.12.1.3 - types.ParsedToken

Questo tipo «opaco» è in realtà un'interfaccia senza nessun metodo. Dato che solo l'*adapter-in* dell'autenticazione può costruire un'istanza di `ParsedToken`, Questo design permette alla *business logic* di operare su di esso senza conoscere la rappresentazione interna dei *token*. Così facendo, in futuro sarebbe possibile cambiare completamente tipologia di *token* senza dover modificare la *business logic*.

3.4.12.1.4 - types.UserData

Questo tipo rappresenta le informazioni di un utente.

Descrizione degli attributi della struttura:

- **`Username string`** - *Username* dell'utente;
- **`Role UserRole`** - Ruolo dell'utente.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.12.1.5 - types.LoginResult

Questo tipo rappresenta il risultato di un'operazione di *login*.

Descrizione degli attributi della struttura:

- **`Token UserToken`** - Un *token* che identifica l'utente;
- **`Role UserRole`** - Il ruolo dell'utente.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.12.1.6 - types.WarehouseOverview

Questo tipo raccoglie le informazioni ad alto livello dei magazzini.

Descrizione degli attributi della struttura:

- **`ID string`** - Identificatore del magazzino.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.12.2 - *Business logic*

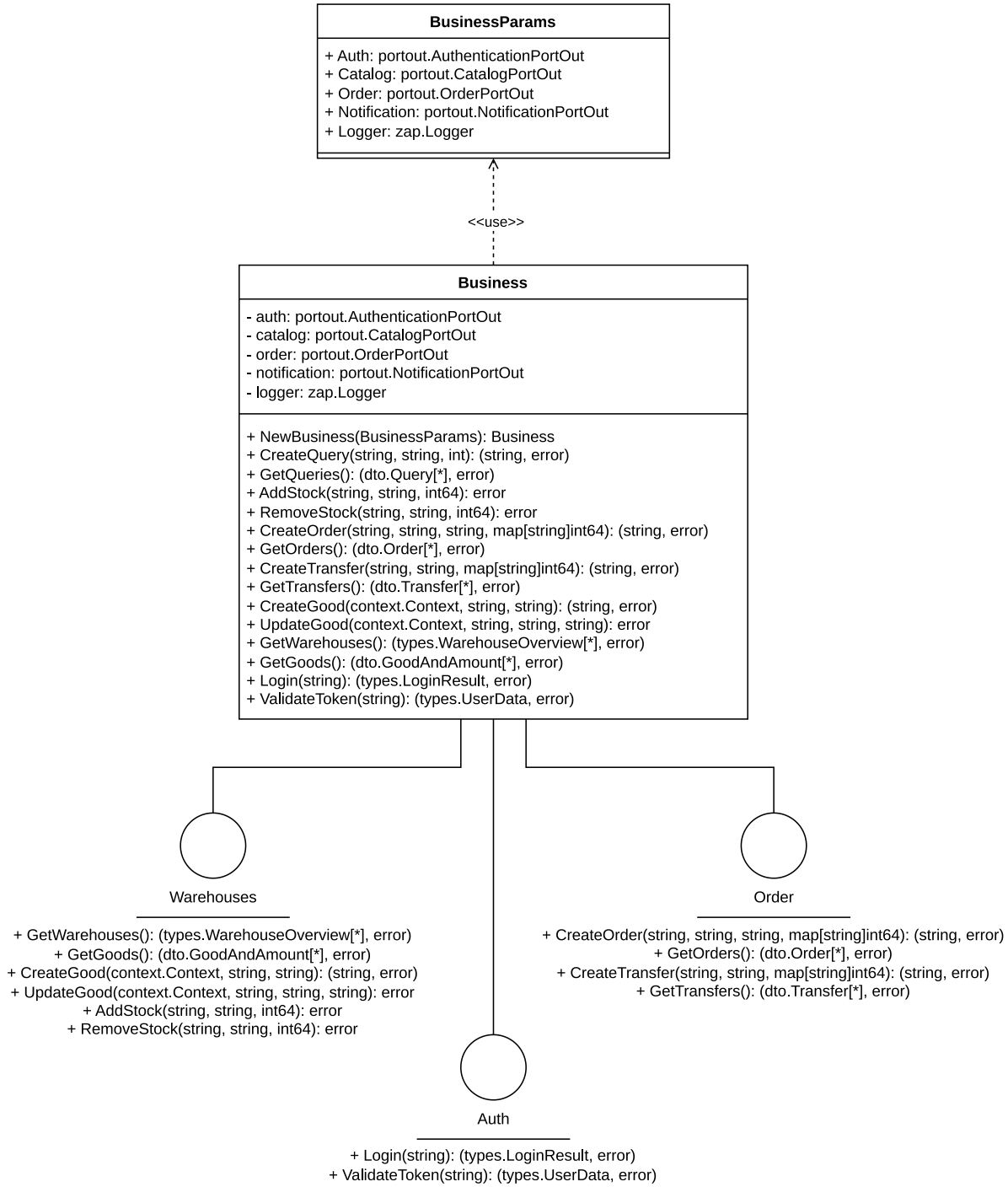

Figura 149: *Business logic* del microservizio API Gateway

In questa sezione sono descritti i tipi relativi alla *business logic*, e le *port-in* da essa implementate.

3.4.12.2.1 - BusinessParams

Parametri in ingresso necessari per costruire un'istanza di gateway_Business.

Descrizione degli attributi della struttura:

- **`Auth portout.AuthenticationPortOut`** - La *Port-out* che verrà utilizzata dalla *business logic* per parlare con altri componenti;

- **`Catalog portout.CatalogPortOut`** - La *Port-out* che verrà utilizzata dalla *business logic* per parlare con altri componenti;
- **`Order portout.OrderPortOut`** - La *Port-out* che verrà utilizzata dalla *business logic* per parlare con altri componenti;
- **`Notification portout.NotificationPortOut`** - La *Port-out* che verrà utilizzata dalla *business logic* per parlare con altri componenti;
- **`Logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.12.2.2 - business.Business

La *business logic*. Coordina tutte le operazioni del microservizio. Implementa le *port-in* Auth, Notifications, Order e Warehouses.

Descrizione degli attributi della struttura:

- **`auth portout.AuthenticationPortOut`** - La *Port-out* che verrà utilizzata per implementare il funzionamento della *business logic*;
- **`catalog portout.CatalogPortOut`** - La *Port-out* che verrà utilizzata per implementare il funzionamento della *business logic*;
- **`order portout.OrderPortOut`** - La *Port-out* che verrà utilizzata per implementare il funzionamento della *business logic*;
- **`notification portout.NotificationPortOut`** - La *Port-out* che verrà utilizzata per implementare il funzionamento della *business logic*;
- **`logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewBusiness(BusinessParams) *Business`** - Costruttore della struttura;
- **`Login(string) (types.LoginResult, error)`** - Dato un username, autentica l'utente;
- **`ValidateToken(string) (types.UserData, error)`** - Valida un *token* fornito dall'utente, per assicurarsi che sia valido;
- **`CreateQuery(string, string, int) (string, error)`** - Crea una regola di notifica;
- **`GetQueries() ([]dto.Query, error)`** - Ritorna le regole di notifica definite;
- **`CreateOrder(string, string, string, map[string]int64) (string, error)`** - Permette di creare un ordine;
- **`GetOrders() ([]dto.Order, error)`** - Ritorna la lista di ordini;
- **`CreateTransfer(string, string, map[string]int64) (string, error)`** - Permette di creare un trasferimento;
- **`GetTransfers() ([]dto.Transfer, error)`** - Ritorna la lista di trasferimenti;
- **`GetWarehouses() ([]types.WarehouseOverview, error)`** - Ritorna le informazioni di tutti i magazzini;
- **`GetGoods() ([]dto.GoodAndAmount, error)`** - Ritorna le informazioni di tutte le merci;
- **`CreateGood(context.Context, string, string) (string, error)`** - Crea una merce;
- **`UpdateGood(context.Context, string, string, string) error`** - Aggiorna le informazioni di una merce;
- **`AddStock(string, string, int64) error`** - Rifornisce un magazzino di una particolare merce;

- **`RemoveStock(string, string, int64) error`** - Estrae una certa quantità di merce da un magazzino.

3.4.12.2.3 - portin.Auth

Questa *port-in* si occupa di gestire le richieste che necessitano di autenticazione.

Descrizione dei metodi dall'interfaccia:

- **`Login(string) (types.LoginResult, error)`** - Dato un username, autentica l'utente;
- **`ValidateToken(string) (types.UserData, error)`** - Valida un *token* fornito dall'utente, per assicurarsi che sia valido.

3.4.12.2.4 - portin.Notifications

Questa *port-in* si occupa di gestire le richieste relative alle notifiche.

Descrizione dei metodi dall'interfaccia:

- **`CreateQuery(string, string, int) (string, error)`** - Crea una regola di notifica;
- **`GetQueries() ([]dto.Query, error)`** - Ritorna le regole di notifica definite.

3.4.12.2.5 - portin.Order

Questa *port-in* si occupa di gestire le richieste relative agli ordini.

Descrizione dei metodi dall'interfaccia:

- **`CreateOrder(string, string, string, map[string]int64) (string, error)`** - Permette di creare un ordine;
- **`GetOrders() ([]dto.Order, error)`** - Ritorna la lista di ordini;
- **`CreateTransfer(string, string, map[string]int64) (string, error)`** - Permette di creare un trasferimento;
- **`GetTransfers() ([]dto.Transfer, error)`** - Ritorna la lista di trasferimenti.

3.4.12.2.6 - portin.Warehouses

Questa *port-in* si occupa di gestire le richieste relative ai magazzini.

Descrizione dei metodi dall'interfaccia:

- **`GetWarehouses() ([]types.WarehouseOverview, error)`** - Ritorna le informazioni di tutti i magazzini;
- **`GetGoods() ([]dto.GoodAndAmount, error)`** - Ritorna le informazioni di tutte le merci;
- **`CreateGood(context.Context, string, string) (string, error)`** - Crea una merce;
- **`UpdateGood(context.Context, string, string, string) error`** - Aggiorna le informazioni di una merce;
- **`AddStock(string, string, int64) error`** - Rifornisce un magazzino di una particolare merce;
- **`RemoveStock(string, string, int64) error`** - Estrae una certa quantità di merce da un magazzino.

3.4.12.3 - *Adapter-in* e *Port-in*

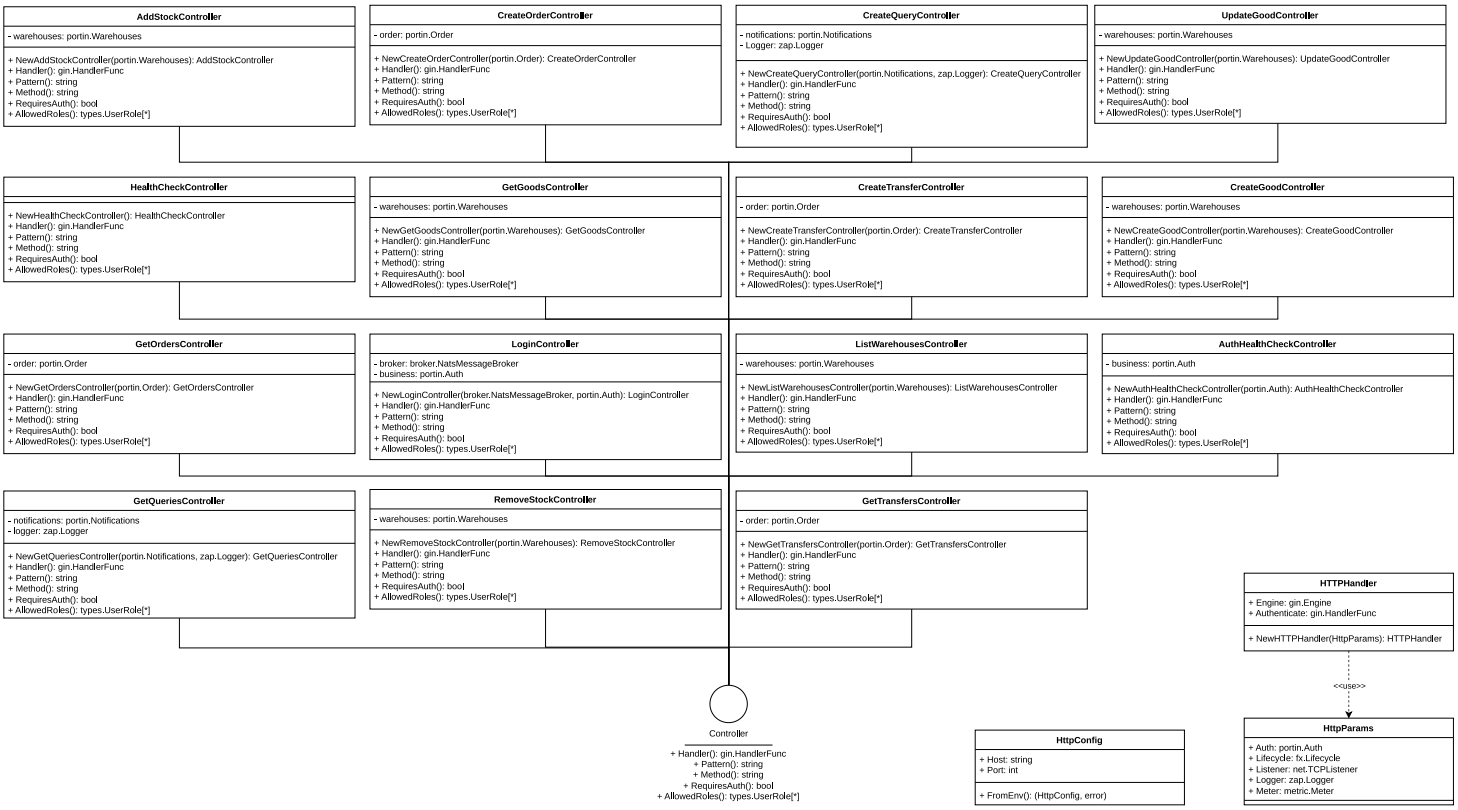

Figura 150: *Adapter-in* di API Gateway

Questi tipi sono stati utilizzati per realizzare la parte di «ingresso» dell'architettura esagonale. Nella sezione corrente sono inclusi i tipi dei *package* `adapterin` e `portin`.

3.4.12.3.1 - adapterin.HttpConfig

Configurazione del server HTTP.

Descrizione degli attributi della struttura:

- **`Host string`** - Indirizzo IP su cui esporre il server HTTP. Utilizzare `0.0.0.0` per esporre su tutti gli indirizzi;
- **`Port int`** - Porta su cui esporre il server HTTP. Esempio: `8080`.

Descrizione dei metodi invocabili dalla struttura:

- **`FromEnv() (*HttpConfig, error)`** - Legge la configurazione dalle variabili d'ambiente.

3.4.12.3.2 - adapterin.HttpParams

Parametri in ingresso necessari per costruire un'istanza di HTTPHandler.

Descrizione degli attributi della struttura:

- **`Auth portin.Auth`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*;
- **`Lifecycle fx.Lifecycle`** - Elemento che consente di eseguire routine di avvio e spegnimento del server HTTP;
- **`Listener *net.TCPListener`** - Un *socket* in ascolto;
- **`Logger *zap.Logger`** - Necessario per raccogliere i *log*;
- **`Meter metric.Meter`** - Necessario per raccogliere le metriche.

Descrizione dei metodi invocabili dalla struttura:

Questa struttura non contiene metodi.

3.4.12.3.3 - adapterin.HTTPHandler

Componente che gestisce le richieste HTTP in ingresso.

Descrizione degli attributi della struttura:

- **`Engine *gin.Engine`** - Componente della libreria HTTP *gin*;
- **`Authenticate gin.HandlerFunc`** - *middleware* che verifica l'autenticazione.

Descrizione dei metodi invocabili dalla struttura:

- **`NewHTTPHandler(HttpParams) *HTTPHandler`** - Costruttore della struttura.

3.4.12.3.4 - adapterin.Controller

Questa interfaccia accomuna tutti i *controller*.

Descrizione dei metodi dall'interfaccia:

- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.5 - adapterin.UpdateGoodController

Questo *adapter* si occupa dell'aggiornamento delle merci.

Descrizione degli attributi della struttura:

- **`warehouses portin.Warehouses`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewUpdateGoodController(portin.Warehouses) *UpdateGoodController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.6 - adapterin.GetTransfersController

Questo *adapter* si occupa di recuperare i trasferimenti.

Descrizione degli attributi della struttura:

- **`order portin.Order`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetTransfersController(portin.Order) *GetTransfersController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.7 - adapterin.RemoveStockController

Questo *adapter* si occupa di rimuovere disponibilità da un magazzino.

Descrizione degli attributi della struttura:

- **`warehouses portin.Warehouses`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewRemoveStockController(portin.Warehouses) *RemoveStockController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.8 - adapterin.GetQueriesController

Questo *adapter* si occupa di recuperare le regole di notifica.

Descrizione degli attributi della struttura:

- **`notifications portin.Notifications`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i *log*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetQueriesController(portin.Notifications, *zap.Logger) *GetQueriesController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.9 - adapterin.GetOrdersController

Questo *adapter* si occupa di recuperare gli ordini.

Descrizione degli attributi della struttura:

- **`order portin.Order`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetOrdersController(portin.Order) *GetOrdersController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.10 - adapterin.LoginController

Questo *adapter* si occupa della procedura di *login*.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`** - Necessario per comunicare con *NATS*.;
- **`business portin.Auth`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewLoginController(*broker.NatsMessageBroker, portin.Auth) *LoginController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;

- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.11 - adapterin.ListWarehousesController

Questo *adapter* si occupa di recuperare la lista di magazzini.

Descrizione degli attributi della struttura:

- **`warehouses portin.Warehouses`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewListWarehousesController(portin.Warehouses) *ListWarehousesController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.12 - adapterin.AuthHealthCheckController

Questo *adapter* si occupa di ritornare all'utente il suo ruolo attuale.

Descrizione degli attributi della struttura:

- **`business portin.Auth`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthHealthCheckController(portin.Auth) *AuthHealthCheckController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.13 - adapterin.HealthCheckController

Questo *adapter* si occupa di ritornare una risposta che segnali il corretto funzionamento del microservizio.

Descrizione degli attributi della struttura:

Questa struttura non contiene attributi.

Descrizione dei metodi invocabili dalla struttura:

- **`NewHealthCheckController() *HealthCheckController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.14 - adapterin.GetGoodsController

Questo *adapter* si occupa di recuperare le merci.

Descrizione degli attributi della struttura:

- **`warehouses portin.Warehouses`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewGetGoodsController(portin.Warehouses) *GetGoodsController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.15 - adapterin.CreateTransferController

Questo *adapter* si occupa della creazione di trasferimenti.

Descrizione degli attributi della struttura:

- **`order portin.Order`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCreateTransferController(portin.Order) *CreateTransferController`** - Costruttore della struttura;

- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.16 - adapterin.CreateQueryController

Questo *adapter* si occupa della creazione di regole di notifica.

Descrizione degli attributi della struttura:

- **`notifications portin.Notifications`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*;
- **`Logger *zap.Logger`** - Necessario per raccogliere i *log*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCreateQueryController(portin.Notifications, *zap.Logger) *CreateQueryController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.17 - adapterin.CreateOrderController

Questo *adapter* si occupa della creazione di ordini.

Descrizione degli attributi della struttura:

- **`order portin.Order`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCreateOrderController(portin.Order) *CreateOrderController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;

- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.18 - adapterin.CreateGoodController

Questo *adapter* si occupa della creazione di merci.

Descrizione degli attributi della struttura:

- **`warehouses portin.Warehouses`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewCreateGoodController(portin.Warehouses) *CreateGoodController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.3.19 - adapterin.AddStockController

Questo *adapter* si occupa dell'aggiunta di disponibilità ad un magazzino.

Descrizione degli attributi della struttura:

- **`warehouses portin.Warehouses`** - Una *port-in* che verrà utilizzata per comunicare con la *business logic*.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAddStockController(portin.Warehouses) *AddStockController`** - Costruttore della struttura;
- **`Handler() gin.HandlerFunc`** - Ritorna una funzione che può gestire le richieste in ingresso;
- **`Pattern() string`** - Ritorna l'*URL* che questo *controller* gestisce;
- **`Method() string`** - Ritorna il metodo che questo *controller* gestisce;
- **`RequiresAuth() bool`** - Ritorna *true* se le richieste effettuate a questo *controller* devono essere autenticate;
- **`AllowedRoles() []types.UserRole`** - Ritorna la lista dei ruoli che possono effettuare richieste a questo *controller*. Se RequresAuth() è falso, questo metodo non verrà mai chiamato.

3.4.12.4 - *Adapter-out* e *Port-out*

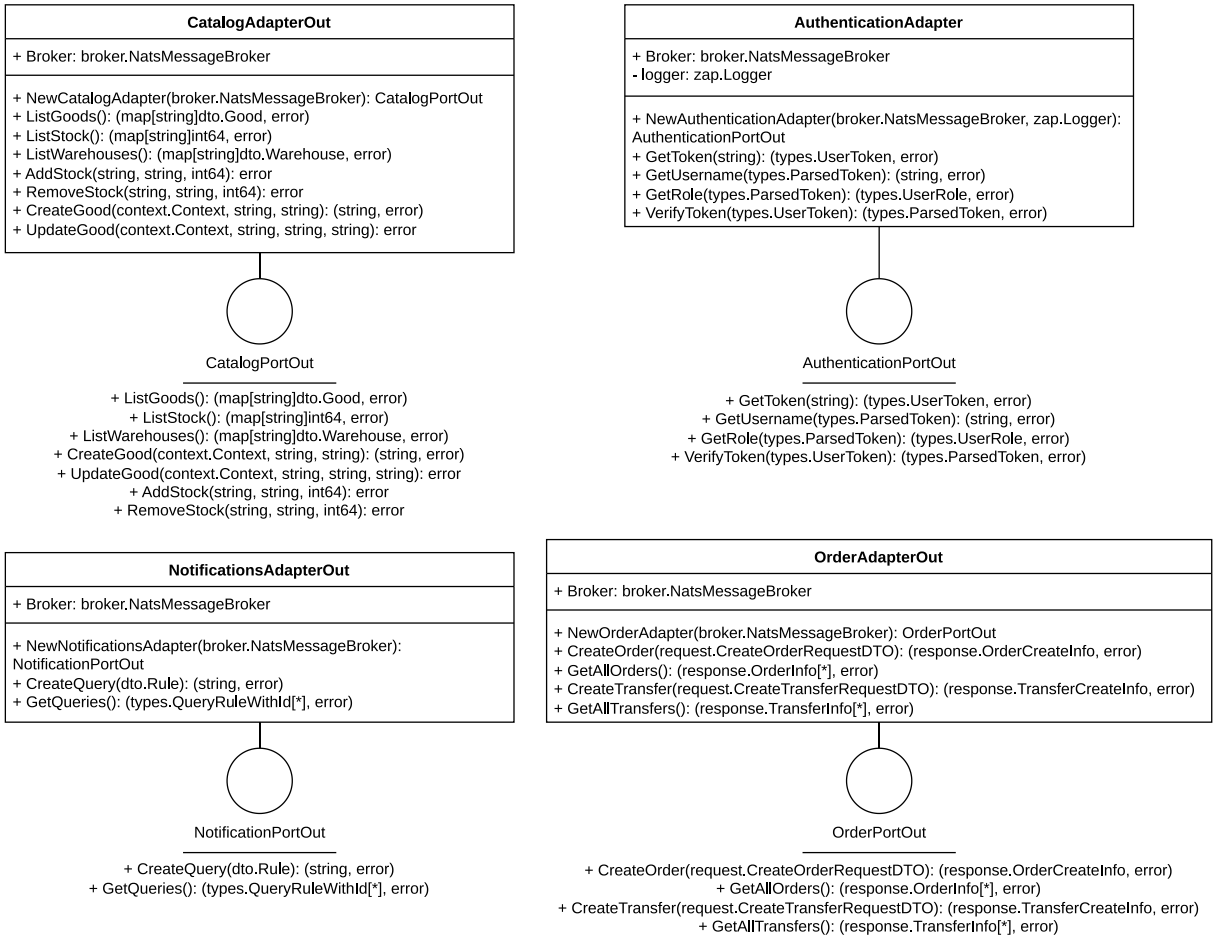

Figura 151: *Adapter-out* di API Gateway

Questi tipi sono stati utilizzati per realizzare la parte di «uscita» dell'architettura esagonale. Nella sezione corrente sono inclusi i tipi dei *package* `adapterout` e `portout`.

3.4.12.4.1 - AuthenticationAdapter

Adapter-out che consente l'interazione con il microservizio di autenticazione.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`** - Necessario per comunicare con *NATS*.;
- **`logger *zap.Logger`** - Necessario per raccogliere i log.

Descrizione dei metodi invocabili dalla struttura:

- **`NewAuthenticationAdapter(broker.NatsMessageBroker, zap.Logger) AuthenticationPortOut`** - Crea una nuova istanza di *AuthenticationAdapter*;
- **`GetToken(string) (types.UserToken, error)`** - Ritorna un *token* per l'*username* fornito, se è valido;
- **`GetUsername(types.ParsedToken) (string, error)`** - Ritorna l'*username* associato al *token* fornito;
- **`GetRole(types.ParsedToken) (types.UserRole, error)`** - Ritorna il ruolo associato al *token* fornito;
- **`VerifyToken(types.UserToken) (types.ParsedToken, error)`** - Verifica che un token sia valido, e recupera i dati ad esso associati.

3.4.12.4.2 - CatalogAdapterOut

Adapter-out che consente l'interazione con il microservizio catalog.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`** - Necessario per comunicare con *NATS*..

Descrizione dei metodi invocabili dalla struttura:

- **`NewCatalogAdapter(broker.NatsMessageBroker) CatalogPortOut`** - Crea una nuova istanza di *CatalogAdapterOut*;
- **`ListGoods() (map[string]dto.Good, error)`** - Consente di recuperare la lista di merci;
- **`ListStock() (map[string]int64, error)`** - Consente di recuperare la disponibilità globale per ogni merce;
- **`ListWarehouses() (map[string]dto.Warehouse, error)`** - Ritorna la lista di magazzini;
- **`AddStock(string, string, int64) error`** - Aggiunge disponibilità ad un magazzino;
- **`RemoveStock(string, string, int64) error`** - Rimuove disponibilità ad un magazzino;
- **`CreateGood(context.Context, string, string) (string, error)`** - Crea una merce;
- **`UpdateGood(context.Context, string, string, string) error`** - Aggiorna una merce.

3.4.12.4.3 - OrderAdapterOut

Adapter-out che consente l'interazione con il microservizio degli ordini.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`** - Necessario per comunicare con *NATS*..

Descrizione dei metodi invocabili dalla struttura:

- **`NewOrderAdapter(broker.NatsMessageBroker) OrderPortOut`** - Crea una nuova istanza di *OrderAdapterOut*;
- **`CreateOrder(request.CreateOrderRequestDTO) (response.OrderCreateInfo, error)`** - Consente di creare un ordine;
- **`GetAllOrders() ([]response.OrderInfo, error)`** - Consente di recuperare la lista di ordini;
- **`CreateTransfer(request.CreateTransferRequestDTO) (response.TransferCreateInfo, error)`** - Consente di creare un trasferimento;
- **`GetAllTransfers() ([]response.TransferInfo, error)`** - Consente di recuperare la lista di trasferimenti.

3.4.12.4.4 - NotificationsAdapterOut

Adapter-out che consente l'interazione con il microservizio delle notifiche.

Descrizione degli attributi della struttura:

- **`broker *broker.NatsMessageBroker`** - Necessario per comunicare con *NATS*..

Descrizione dei metodi invocabili dalla struttura:

- **`NewNotificationsAdapter(broker.NatsMessageBroker) NotificationPortOut`** - Crea una nuova istanza di *NotificationsAdapterOut*;
- **`CreateQuery(dto.Rule) (string, error)`** - Permette di creare una nuova regola;
- **`GetQueries() ([]types.QueryRuleWithId, error)`** - Permette di ottenere la lista di regole.

3.4.12.4.5 - OrderPortOut

Port-out che consente la comunicazione col microservizio di gestione degli ordini.

Descrizione dei metodi dall'interfaccia:

- **`CreateOrder(request.CreateOrderRequestDTO) (response.OrderCreateInfo, error)`** - Consente di creare un ordine;
- **`GetAllOrders() ([]response.OrderInfo, error)`** - Consente di recuperare la lista di ordini;
- **`CreateTransfer(request.CreateTransferRequestDTO) (response.TransferCreateInfo, error)`** - Consente di creare un trasferimento;
- **`GetAllTransfers() ([]response.TransferInfo, error)`** - Consente di recuperare la lista di trasferimenti.

3.4.12.4.6 - CatalogPortOut

Port-out che consente la comunicazione col microservizio di gestione del catalogo.

Descrizione dei metodi dall'interfaccia:

- **`ListGoods() (map[string]dto.Good, error)`** - Consente di recuperare la lista di merci;
- **`ListStock() (map[string]int64, error)`** - Consente di recuperare la disponibilità globale per ogni merce;
- **`ListWarehouses() (map[string]dto.Warehouse, error)`** - Ritorna la lista di magazzini;
- **`AddStock(string, string, int64) error`** - Aggiunge disponibilità ad un magazzino;
- **`RemoveStock(string, string, int64) error`** - Rimuove disponibilità ad un magazzino;
- **`CreateGood(context.Context, string, string) (string, error)`** - Crea una merce;
- **`UpdateGood(context.Context, string, string, string) error`** - Aggiorna una merce.

3.4.12.4.7 - NotificationPortOut

Port-out che consente la comunicazione col microservizio di notifica.

Descrizione dei metodi dall'interfaccia:

- **`CreateQuery(dto.Rule) (string, error)`** - Permette di creare una nuova regola;
- **`GetQueries() ([]types.QueryRuleWithId, error)`** - Permette di ottenere la lista di regole.

3.4.12.4.8 - AuthenticationPortOut

Port-out che consente la comunicazione col microservizio di autenticazione.

Descrizione dei metodi dall'interfaccia:

- **`GetToken(string) (types.UserToken, error)`** - Ritorna un *token* per l'*username* fornito, se è valido;
- **`GetUsername(types.ParsedToken) (string, error)`** - Ritorna l'*username* associato al *token* fornito;
- **`GetRole(types.ParsedToken) (types.UserRole, error)`** - Ritorna il ruolo associato al *token* fornito;
- **`VerifyToken(types.UserToken) (types.ParsedToken, error)`** - Verifica che un token sia valido, e recupera i dati ad esso associati.

4 - Stato dei requisiti funzionali

4.1 - Stato per requisito

Codice	Descrizione	Stato
R-1-F-Ob	L'Utente deve poter autenticarsi presso il Sistema	Soddisfatto
R-2-F-Ob	L'Utente deve inserire la tipologia di utente (Cliente, Admin Globale o Admin Locale) per potersi autenticare al Sistema	Soddisfatto
R-3-F-De	L'Utente deve inserire il proprio Username per potersi autenticare	Non Soddisfatto
R-4-F-De	L'Utente deve inserire la propria Password per potersi autenticare	Non Soddisfatto
R-5-F-Ob	L'Utente deve ricevere un errore in seguito ad un tentativo di accesso/autenticazione non riuscito	Non Soddisfatto
R-6-F-Ob	Il Cliente deve poter creare un ordine che può confermare in seguito.	Soddisfatto
R-7-F-Ob	Il Cliente deve inserire il nome dell'ordine al momento della creazione di un nuovo ordine da confermare	Soddisfatto
R-8-F-Ob	Il Cliente deve inserire il nominativo del destinatario dell'ordine al momento della creazione di un nuovo ordine da confermare	Soddisfatto
R-9-F-Ob	Il Cliente deve inserire l'indirizzo di spedizione dell'ordine al momento della creazione di un nuovo ordine da confermare	Soddisfatto
R-10-F-Ob	Il Cliente deve poter aggiungere merce ad un ordine non ancora confermato, indipendentemente se si tratta di un ordine di natura locale (limitata al magazzino corrente) o globale	Soddisfatto
R-11-F-Ob	Il Cliente, durante l'operazione di aggiunta merce ad un ordine non ancora confermato, deve selezionare la merce che vuole aggiungere ad un ordine non ancora confermato	Soddisfatto
R-12-F-Ob	Il Cliente, durante l'operazione di aggiunta merce ad un ordine non ancora confermato, deve inserire il quantitativo della merce che vuole aggiungere ad un ordine non ancora confermato	Soddisfatto
R-13-F-Ob	Il Cliente, durante l'operazione di aggiunta merce ad un ordine non ancora confermato, deve selezionare	Soddisfatto

Codice	Descrizione	Stato
	l'ordine non ancora confermato alla quale vuole aggiungere la merce	
R-14-F-Ob	Il Cliente deve ricevere un errore qualora la merce aggiunta all'ordine non risulti essere valida (ovvero la quantità della merce è insufficiente oppure la merce non esiste)	Soddisfatto
R-15-F-Ob	Il Cliente deve ricevere un errore quando sta cercando di fare un'operazione su un ordine non confermato (quale l'aggiunta di merce, la cancellazione od una conferma) ma nessun ordine non confermato è disponibile	Soddisfatto
R-16-F-Ob	Il Cliente deve poter cancellare un ordine non ancora confermato, selezionando quale ordine cancellare	Soddisfatto
R-17-F-Ob	Il Cliente deve poter confermare un ordine non ancora confermato, selezionando quale ordine cancellare	Soddisfatto
R-18-F-Ob	Il Cliente deve poter visualizzare l'elenco degli ordini non confermati per l'utente attualmente autenticato	Soddisfatto
R-19-F-Ob	Il Cliente deve poter visualizzare l'ID di ciascun ordine nella lista degli ordini non confermati	Soddisfatto
R-20-F-Ob	Il Cliente deve poter visualizzare la data di creazione di ciascun ordine nella lista degli ordini non confermati	Soddisfatto
R-21-F-Ob	Il Cliente deve poter visualizzare il nome di ciascun ordine nella lista degli ordini non confermati	Soddisfatto
R-22-F-Ob	Il Cliente deve poter consultare i dettagli di un ordine non confermato	Soddisfatto
R-23-F-Ob	Il Cliente, visualizzando un ordine non confermato nel dettaglio, deve visualizzarne l'ID	Soddisfatto
R-24-F-Ob	Il Cliente, visualizzando un ordine non confermato nel dettaglio, deve visualizzarne la data di creazione	Soddisfatto
R-25-F-Ob	Il Cliente, visualizzando un ordine non confermato nel dettaglio, deve visualizzarne il nome	Soddisfatto
R-26-F-Ob	Il Cliente, visualizzando un ordine non confermato nel dettaglio, deve visualizzarne la lista delle merci	Soddisfatto
R-27-F-Ob	Il Cliente, visualizzando l'elenco merce di un ordine non confermato, deve poter visualizzare la quantità della merce	Soddisfatto

Codice	Descrizione	Stato
R-28-F-Ob	Il Cliente, visualizzando l'elenco merce di un ordine non confermato, deve poter visualizzare il nome della merce	Soddisfatto
R-29-F-Ob	Il Cliente deve poter visualizzare la lista delle merci nel Sistema	Soddisfatto
R-30-F-Ob	Il Cliente visualizzando la lista delle merci nel Sistema, deve poter visualizzare l'ID di ciascuna delle merci	Soddisfatto
R-31-F-Ob	Il Cliente visualizzando la lista delle merci nel Sistema, deve poter visualizzare il nome di ciascuna delle merci	Soddisfatto
R-32-F-Ob	Il Cliente visualizzando la lista delle merci nel Sistema, deve poter visualizzare la quantità della merce complessiva in tutti i magazzini di ciascuna delle merci	Soddisfatto
R-33-F-Ob	Il Cliente visualizzando la lista delle merci nel Sistema, deve poter visualizzare la quantità della merce attualmente presente nel magazzino di ciascuna delle merci	Soddisfatto
R-34-F-Ob	Il Cliente deve poter visualizzare una merce nel Sistema nel dettaglio	Soddisfatto
R-35-F-Ob	Il Cliente visualizzando una merce specifica nel Sistema, deve poter visualizzare l'ID di tale merce	Soddisfatto
R-36-F-Ob	Il Cliente visualizzando una merce specifica nel Sistema, deve poter visualizzare il nome di tale merce	Soddisfatto
R-37-F-Ob	Il Cliente visualizzando una merce specifica nel Sistema, deve poter visualizzare la quantità della merce complessiva in tutti i magazzini di tale merce	Soddisfatto
R-38-F-Ob	Il Cliente visualizzando una merce specifica nel Sistema, deve poter visualizzare la quantità della merce attualmente presente nel magazzino di tale merce	Soddisfatto
R-39-F-Ob	Il Cliente visualizzando una merce specifica nel Sistema, deve poter visualizzare la descrizione di tale merce	Soddisfatto
R-40-F-Ob	L'Admin Globale deve poter creare un trasferimento da confermare in seguito.	Soddisfatto
R-41-F-Ob	L'Admin Globale, durante la creazione di un trasferimento da confermare in seguito, deve selezionare il magazzino mittente	Soddisfatto

Codice	Descrizione	Stato
R-42-F-Ob	L'Admin Globale, durante la creazione di un trasferimento da confermare in seguito, deve selezionare il magazzino destinatario	Soddisfatto
R-43-F-Ob	L'Admin Globale deve poter aggiungere della merce ad un trasferimento non confermato	Soddisfatto
R-44-F-Ob	L'Admin Globale, durante l'operazione di aggiunta di merce ad un trasferimento non confermato, deve selezionare la merce da aggiungere	Soddisfatto
R-45-F-Ob	L'Admin Globale, durante l'operazione di aggiunta di merce ad un trasferimento non confermato, deve selezionare la quantità di merce da aggiungere	Soddisfatto
R-46-F-Ob	L'Admin Globale, durante l'operazione di aggiunta di merce ad un trasferimento non confermato, deve selezionare il trasferimento non confermato alla quale aggiungere la merce	Soddisfatto
R-47-F-Ob	L'Admin Globale deve poter confermare un trasferimento non ancora confermato, selezionando quale trasferimento confermare	Soddisfatto
R-48-F-Ob	L'Admin Globale deve ricevere un errore se la merce in un trasferimento che vuole confermare non è più disponibile in quantità sufficiente o non è più esistente nel Sistema	Soddisfatto
R-49-F-Ob	L'Admin Globale deve ricevere un errore qualora selezioni di voler aggiungere merce, confermare o cancellare un trasferimento non confermato ma nessun trasferimento non confermato risulta essere presente	Soddisfatto
R-50-F-Ob	L'Admin Globale deve poter cancellare un trasferimento non ancora confermato, selezionando quale trasferimento cancellare	Soddisfatto
R-51-F-Ob	L'Admin Globale deve poter visualizzare l'elenco di tutti i trasferimenti	Soddisfatto
R-52-F-Ob	L'Admin Globale deve poter visualizzare per ogni trasferimento dell'elenco di tutti i trasferimenti, l'ID del trasferimento	Soddisfatto
R-53-F-Ob	L'Admin Globale deve poter visualizzare per ogni trasferimento dell'elenco di tutti i trasferimenti, lo stato del trasferimento	Soddisfatto

Codice	Descrizione	Stato
R-54-F-Ob	L'Admin Globale deve poter visualizzare un singolo trasferimento nello specifico	Soddisfatto
R-55-F-Ob	L'Admin Globale, visualizzando un singolo trasferimento, deve visualizzare l'ID del trasferimento	Soddisfatto
R-56-F-Ob	L'Admin Globale, visualizzando un singolo trasferimento, deve visualizzare il magazzino mittente del trasferimento	Soddisfatto
R-57-F-Ob	L'Admin Globale, visualizzando un singolo trasferimento, deve visualizzare il magazzino di destinazione del trasferimento	Soddisfatto
R-58-F-Ob	L'Admin Globale, visualizzando un singolo trasferimento, deve visualizzare lo stato del trasferimento	Soddisfatto
R-59-F-Ob	L'Admin Globale, visualizzando un singolo trasferimento, deve visualizzare l'elenco della merce interessata dal trasferimento	Soddisfatto
R-60-F-Ob	L'Admin Globale, visualizzando un trasferimento nel dettaglio, deve visualizzare per ogni merce interessata il nome di tale merce	Soddisfatto
R-61-F-Ob	L'Admin Globale, visualizzando un trasferimento nel dettaglio, deve visualizzare per ogni merce interessata la quantità di tale merce	Soddisfatto
R-62-F-Ob	L'Admin Globale deve poter visualizzare l'elenco delle notifiche contenenti i consigli di rifornimento	Soddisfatto
R-63-F-Ob	L'Admin Globale deve visualizzare, per ogni notifica nell'elenco delle notifiche di rifornimento, l'ID della notifica	Soddisfatto
R-64-F-Ob	L'Admin Globale deve visualizzare, per ogni notifica nell'elenco delle notifiche di rifornimento, lo stato della notifica (confermato, da confermare, rifiutato)	Soddisfatto
R-65-F-Ob	L'Admin Globale deve ricevere un messaggio di errore quando tenta di compiere un'azione sulle notifiche di rifornimento, ma nessuna notifica è disponibile	Soddisfatto
R-66-F-De	L'Admin Globale deve poter visualizzare le notifiche di rifornimento suggerite da *Machine Learning*	Non Soddisfatto

Codice	Descrizione	Stato
R-67-F-De	L'Admin Globale deve visualizzare, per ogni notifica nell'elenco delle notifiche di rifornimento da parte di *Machine Learning*, l'ID della notifica	Non Soddisfatto
R-68-F-De	L'Admin Globale deve visualizzare, per ogni notifica nell'elenco delle notifiche di rifornimento da parte di un *Machine Learning*, lo stato della notifica (confermato, da confermare, rifiutato)	Non Soddisfatto
R-69-F-Ob	L'Admin Globale deve poter visualizzare una notifica di rifornimento nello specifico	Soddisfatto
R-70-F-Ob	L'Admin Globale, visualizzando una notifica di rifornimento nello specifico, deve visualizzarne l'ID	Soddisfatto
R-71-F-Ob	L'Admin Globale, visualizzando una notifica di rifornimento nello specifico, deve visualizzarne lo stato (confermato, da confermare, rifiutato)	Soddisfatto
R-72-F-Ob	L'Admin Globale, visualizzando una notifica di rifornimento nello specifico, deve visualizzarne il magazzino destinatario	Soddisfatto
R-73-F-Ob	L'Admin Globale, visualizzando una notifica di rifornimento nello specifico, deve visualizzarne l'elenco della merce	Soddisfatto
R-74-F-Ob	Per ciascuna merce il cui rifornimento è consigliato da una notifica di rifornimento, l'Admin Globale deve visualizzare, quando sta visualizzando una notifica in particolare, l'ID della merce	Soddisfatto
R-75-F-Ob	Per ciascuna merce il cui rifornimento è consigliato da una notifica di rifornimento, l'Admin Globale deve visualizzare, quando sta visualizzando una notifica in particolare, il nome della merce	Soddisfatto
R-76-F-Ob	Per ciascuna merce il cui rifornimento è consigliato da una notifica di rifornimento, l'Admin Globale deve visualizzare, quando sta visualizzando una notifica in particolare, la quantità della merce da rifornire	Soddisfatto
R-77-F-Ob	L'Admin Globale deve poter accettare una notifica di rifornimento non ancora accettata, selezionando quale accettare	Soddisfatto
R-78-F-Ob	L'Admin Globale deve poter rifiutare una notifica di rifornimento non ancora accettata, selezionando quale rifiutare	Soddisfatto

Codice	Descrizione	Stato
R-79-F-Ob	L'Admin Globale deve poter visualizzare l'elenco dei microservizi	Soddisfatto
R-80-F-Ob	L'Admin Globale, visualizzando l'elenco dei microservizi, deve visualizzare il nome di ciascun microservizio	Soddisfatto
R-81-F-Ob	L'Admin Globale, visualizzando l'elenco dei microservizi, deve visualizzare il numero di richieste al secondo di ciascun microservizio	Soddisfatto
R-82-F-De	L'Admin Globale deve poter esportare gli ordini eseguiti su un file di tipo *.csv*	Non Soddisfatto
R-83-F-De	L'Admin Globale deve ricevere un errore quando tenta di esportare degli ordini in un file in formato *.csv* ma nessun ordine da esportare è presente	Non Soddisfatto
R-84-F-De	L'Admin Globale deve poter esportare il report dell'inventario globale in un file in formato *.csv*	Non Soddisfatto
R-85-F-De	L'Admin Globale deve ricevere un errore quando cerca di esportare l'inventario ma nessun dato è disponibile	Non Soddisfatto
R-86-F-Ob	L'Admin Globale deve poter impostare una soglia minima di allerta per una merce	Soddisfatto
R-87-F-Ob	L'Admin Globale, impostando una soglia minima di allerta, deve selezionare la merce a cui assegnare la nuova soglia	Soddisfatto
R-88-F-Ob	L'Admin Globale, impostando una soglia minima di allerta, deve inserire la nuova soglia	Soddisfatto
R-89-F-Ob	L'Admin Globale deve ricevere un errore se la soglia minima di allerta che ha impostato non è valida (ad esempio perché negativa)	Soddisfatto
R-90-F-Ob	L'Admin Locale deve poter manualmente aggiungere stock (quantità) di merce ad una merce esistente nel Sistema	Soddisfatto
R-91-F-Ob	L'Admin Locale, aggiungendo uno stock di merce, deve selezionare la merce a cui aggiungere lo stock	Soddisfatto
R-92-F-Ob	L'Admin Locale, aggiungendo uno stock di merce, deve inserire la quantità da aggiungere	Soddisfatto
R-93-F-Ob	L'Admin Globale deve poter creare (aggiungere) una merce nel Sistema	Soddisfatto

Codice	Descrizione	Stato
R-94-F-Ob	L'Admin Globale, creando (aggiungendo) una merce al Sistema, deve indicare il nome	Soddisfatto
R-95-F-Ob	L'Admin Globale, creando (aggiungendo) una merce al Sistema, deve indicare la descrizione	Soddisfatto
R-96-F-Ob	L'Admin Globale deve poter aggiornare le informazioni di una merce	Soddisfatto
R-97-F-Ob	L'Admin Globale, modificando una merce del Sistema, deve indicare quale merce modificare	Soddisfatto
R-98-F-Ob	L'Admin Globale, modificando una merce del Sistema, deve indicare il nome	Soddisfatto
R-99-F-Ob	L'Admin Globale, modificando una merce del Sistema, deve indicare la descrizione	Soddisfatto
R-100-F-Ob	Il Cliente deve poter visualizzare l'elenco degli ordini eseguiti	Soddisfatto
R-101-F-Ob	Il Cliente, per ciascun ordine nell'elenco degli ordini eseguiti, deve visualizzarne l'ID	Soddisfatto
R-102-F-Ob	Il Cliente, per ciascun ordine nell'elenco degli ordini eseguiti, deve visualizzarne la data di creazione	Soddisfatto
R-103-F-Ob	Il Cliente, per ciascun ordine nell'elenco degli ordini eseguiti, deve visualizzarne il nome	Soddisfatto
R-104-F-Ob	Il Cliente deve poter visualizzare il dettaglio di un ordine eseguito	Soddisfatto
R-105-F-Ob	Il Cliente, visualizzando un ordine eseguito nel dettaglio, deve visualizzarne l'ID	Soddisfatto
R-106-F-Ob	Il Cliente, visualizzando un ordine eseguito nel dettaglio, deve visualizzarne la data di creazione	Soddisfatto
R-107-F-Ob	Il Cliente, visualizzando un ordine eseguito nel dettaglio, deve visualizzarne il nome	Soddisfatto
R-108-F-Ob	Il Cliente, visualizzando un ordine eseguito nel dettaglio, deve visualizzarne la lista delle merci	Soddisfatto
R-109-F-Ob	Per ogni merce nella lista delle merci di un ordine eseguito, il Cliente deve visualizzare il nome della merce	Soddisfatto

Codice	Descrizione	Stato
R-110-F-Ob	Per ogni merce nella lista delle merci di un ordine eseguito, il Cliente deve visualizzare la quantità interessata dall'ordine	Soddisfatto
R-111-F-De	L'Admin Locale deve avere la possibilità di creare un Backup del proprio magazzino	Non Soddisfatto
R-112-F-De	L'Admin Locale deve avere la possibilità di attivare un Backup periodico del proprio magazzino, selezionandone la periodicità	Non Soddisfatto
R-113-F-De	L'Admin Locale deve ricevere un errore se la periodicità del Backup periodico che ha selezionato non è valida	Non Soddisfatto
R-114-F-De	L'Admin Locale deve poter eliminare la realizzazione del Backup periodico	Non Soddisfatto
R-115-F-De	L'Admin Locale deve ricevere un errore quando vuole eliminare la realizzazione di un Backup periodico ma non è attivo un Backup periodico	Non Soddisfatto
R-116-F-De	L'Admin Locale deve avere la possibilità di ripristinare i dati dell'ultimo Backup effettuato	Non Soddisfatto
R-117-F-De	L'Admin Locale deve ricevere un errore quando vuole ripristinare i dati dell'ultimo Backup effettuato ma nessun Backup è presente	Non Soddisfatto
R-118-F-De	L'Admin Globale deve poter visualizzare l'elenco delle attività di accesso	Non Soddisfatto
R-119-F-De	L'Admin Globale, visualizzando l'elenco delle attività di accesso, deve visualizzare l'Indirizzo IP del luogo di accesso di ciascuna	Non Soddisfatto
R-120-F-De	L'Admin Globale, visualizzando l'elenco delle attività di accesso, deve visualizzare l'ID di ciascuna	Non Soddisfatto
R-121-F-De	L'Admin Globale, visualizzando l'elenco delle attività di accesso, deve visualizzare lo stato di ciascuna (riuscito, bloccato o negato)	Non Soddisfatto
R-122-F-De	L'Admin Globale deve poter bloccare un tentativo di accesso, bloccando l'indirizzo IP dalla quale questo è avvenuto, inserendo l'ID del tentativo	Non Soddisfatto
R-123-F-De	Il Sistema di rilevamento deve notificare via email/sms gli Admin globali eventi di opportuna importanza, quali	Non Soddisfatto

Codice	Descrizione	Stato
	il raggiungimento di scorte minime o la necessità di approvare un rifornimento	
R-124-F-De	L'Admin Globale deve avere la possibilità di aggiungere un Utente al Sistema	Non Soddisfatto
R-125-F-De	L'Admin Globale, aggiungendo un nuovo Utente, deve inserirne il nome	Non Soddisfatto
R-126-F-De	L'Admin Globale, aggiungendo un nuovo Utente, deve inserirne la Password	Non Soddisfatto
R-127-F-De	L'Admin Globale, aggiungendo un nuovo Utente, deve inserirne il ruolo	Non Soddisfatto
R-128-F-De	L'Admin Globale deve poter eliminare un Utente dal Sistema, selezionando quale Utente	Non Soddisfatto
R-129-F-De	L'Admin Globale deve poter promuovere il ruolo di un Utente, selezionando quale Utente	Non Soddisfatto
R-130-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione dell'elenco delle merci disponibili	Soddisfatto
R-131-F-Ob	Lo Scheduler, avviando la sincronizzazione delle merci disponibili, deve avviare la sincronizzazione della quantità localmente disponibile per ciascuna merce	Soddisfatto
R-132-F-Ob	Lo Scheduler, avviando la sincronizzazione delle merci disponibili, deve avviare la sincronizzazione della quantità globalmente disponibile per ciascuna merce	Soddisfatto
R-133-F-Ob	Lo Scheduler, avviando la sincronizzazione delle merci disponibili, deve avviare la sincronizzazione del nome per ciascuna merce	Soddisfatto
R-134-F-Ob	Lo Scheduler, avviando la sincronizzazione delle merci disponibili, deve avviare la sincronizzazione della descrizione per ciascuna merce	Soddisfatto
R-135-F-Ob	Lo Scheduler, avviando la sincronizzazione delle merci disponibili, deve avviare la sincronizzazione dell'ID per ciascuna merce	Soddisfatto
R-136-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione di una nuova merce	Soddisfatto
R-137-F-Ob	Lo Scheduler, avviando la sincronizzazione della merce aggiunta, deve avviare la sincronizzazione della quantità localmente disponibile	Soddisfatto

Codice	Descrizione	Stato
R-138-F-Ob	Lo Scheduler, avviando la sincronizzazione della merce aggiunta, deve avviare la sincronizzazione della quantità globalmente disponibile	Soddisfatto
R-139-F-Ob	Lo Scheduler, avviando la sincronizzazione della merce aggiunta, deve avviare la sincronizzazione del nome	Soddisfatto
R-140-F-Ob	Lo Scheduler, avviando la sincronizzazione della merce aggiunta, deve avviare la sincronizzazione della descrizione	Soddisfatto
R-141-F-Ob	Lo Scheduler, avviando la sincronizzazione della merce aggiunta, deve avviare la sincronizzazione dell'ID	Soddisfatto
R-142-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione di una merce eliminata	Soddisfatto
R-143-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce eliminata, deve avviare la sincronizzazione della quantità localmente disponibile	Soddisfatto
R-144-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce eliminata, deve avviare la sincronizzazione della quantità globalmente disponibile	Soddisfatto
R-145-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce eliminata, deve avviare la sincronizzazione del nome	Soddisfatto
R-146-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce eliminata, deve avviare la sincronizzazione della descrizione	Soddisfatto
R-147-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce eliminata, deve avviare la sincronizzazione dell'ID	Soddisfatto
R-148-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione di una merce modificata	Soddisfatto
R-149-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce modificata, deve avviare la sincronizzazione della quantità localmente disponibile	Soddisfatto
R-150-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce modificata, deve avviare la sincronizzazione della quantità globalmente disponibile	Soddisfatto

Codice	Descrizione	Stato
R-151-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce modificata, deve avviare la sincronizzazione del nome	Soddisfatto
R-152-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce modificata, deve avviare la sincronizzazione della descrizione	Soddisfatto
R-153-F-Ob	Lo Scheduler, avviando la sincronizzazione di una merce modificata, deve avviare la sincronizzazione dell'ID	Soddisfatto
R-154-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione dell'elenco degli ordini	Soddisfatto
R-155-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini, deve sincronizzare per ciascun ordine la data di creazione	Soddisfatto
R-156-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini, deve sincronizzare per ciascun ordine il nome	Soddisfatto
R-157-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini, deve sincronizzare per ciascun ordine l'ID	Soddisfatto
R-158-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini, deve sincronizzare per ciascun ordine lo stato	Soddisfatto
R-159-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini, deve sincronizzare per ciascun ordine la lista delle merci interessate	Soddisfatto
R-160-F-Ob	Per ogni merce di un ordine da sincronizzare, lo Scheduler deve avviare la sincronizzazione dell'ID della merce	Soddisfatto
R-161-F-Ob	Per ogni merce di un ordine da sincronizzare, lo Scheduler deve avviare la sincronizzazione la quantità interessata della merce	Soddisfatto
R-162-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione degli ordini confermati	Soddisfatto
R-163-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini confermati, avvia la sincronizzazione di tutti gli ordini	Soddisfatto
R-164-F-Ob	Lo Scheduler deve poter avviare la sincronizzazione degli ordini cancellati	Soddisfatto

Codice	Descrizione	Stato
R-165-F-Ob	Lo Scheduler, avviando la sincronizzazione degli ordini cancellati, avvia la sincronizzazione di tutti gli ordini	Soddisfatto
R-166-F-Ob	Lo Scheduler deve avviare la sincronizzazione dell'elenco dei trasferimenti	Soddisfatto
R-167-F-Ob	Lo Scheduler, avviando la sincronizzazione dell'elenco dei trasferimenti, deve sincronizzare il magazzino destinatario di ciascun trasferimento	Soddisfatto
R-168-F-Ob	Lo Scheduler, avviando la sincronizzazione dell'elenco dei trasferimenti, deve sincronizzare il magazzino mittente di ciascun trasferimento	Soddisfatto
R-169-F-Ob	Lo Scheduler, avviando la sincronizzazione dell'elenco dei trasferimenti, deve sincronizzare l'ID di ciascun trasferimento	Soddisfatto
R-170-F-Ob	Lo Scheduler, avviando la sincronizzazione dell'elenco dei trasferimenti, deve sincronizzare lo stato di ciascun trasferimento	Soddisfatto
R-171-F-Ob	Lo Scheduler, avviando la sincronizzazione dell'elenco dei trasferimenti, deve sincronizzare la lista delle merci di ciascun trasferimento	Soddisfatto
R-172-F-Ob	Per ogni merce di un trasferimento da sincronizzare, lo Scheduler deve avviare la sincronizzazione dell'ID della merce	Soddisfatto
R-173-F-Ob	Per ogni merce di un trasferimento da sincronizzare, lo Scheduler deve avviare la sincronizzazione la quantità interessata della merce	Soddisfatto
R-174-F-Ob	Lo Scheduler deve avviare la sincronizzazione dei trasferimenti confermati	Soddisfatto
R-175-F-Ob	Lo Scheduler, avviando la sincronizzazione dei trasferimenti confermati, deve avviare la sincronizzazione di tutti i trasferimenti	Soddisfatto
R-176-F-Ob	Lo Scheduler deve avviare la sincronizzazione dei trasferimenti cancellati	Soddisfatto
R-177-F-Ob	Lo Scheduler, avviando la sincronizzazione dei trasferimenti cancellati, deve avviare la sincronizzazione di tutti i trasferimenti	Soddisfatto
R-178-F-Ob	Lo Scheduler deve avviare la sincronizzazione delle notifiche di rifornimento	Soddisfatto

Codice	Descrizione	Stato
R-179-F-Ob	Lo Scheduler, avviando la sincronizzazione delle notifiche di rifornimento, deve sincronizzare per ciascuna notifica l'ID	Soddisfatto
R-180-F-Ob	Lo Scheduler, avviando la sincronizzazione delle notifiche di rifornimento, deve sincronizzare per ciascuna notifica il magazzino destinatario	Soddisfatto
R-181-F-Ob	Lo Scheduler, avviando la sincronizzazione delle notifiche di rifornimento, deve sincronizzare per ciascuna notifica lo stato	Soddisfatto
R-182-F-Ob	Lo Scheduler, avviando la sincronizzazione delle notifiche di rifornimento, deve sincronizzare per ciascuna notifica l'elenco delle merci interessate	Soddisfatto
R-183-F-Ob	Per ogni merce facente parte di un elenco merci di un trasferimento, lo Scheduler deve sincronizzare l'ID della merce	Soddisfatto
R-184-F-Ob	Per ogni merce facente parte di un elenco merci di un trasferimento, lo Scheduler deve sincronizzare la quantità interessata della merce	Soddisfatto
R-185-F-Ob	Lo Scheduler deve avviare la sincronizzazione delle notifiche di rifornimento confermate	Soddisfatto
R-186-F-Ob	Lo Scheduler, avviando la sincronizzazione delle notifiche di rifornimento confermate, deve sincronizzare tutte le notifiche di rifornimento	Soddisfatto
R-187-F-Ob	Lo Scheduler deve avviare la sincronizzazione delle notifiche di rifornimento cancellate	Soddisfatto
R-188-F-Ob	Lo Scheduler, avviando la sincronizzazione delle notifiche di rifornimento cancellate, deve sincronizzare tutte le notifiche di rifornimento	Soddisfatto
R-189-F-Ob	Lo Scheduler deve avviare la sincronizzazione dei dati dei microservizi	Soddisfatto
R-190-F-Ob	Lo Scheduler, avviando la sincronizzazione dei dati dei microservizi, deve avviare la sincronizzazione del nome di ciascun microservizio	Soddisfatto
R-191-F-Ob	Lo Scheduler, avviando la sincronizzazione dei dati dei microservizi, deve avviare la sincronizzazione delle richieste al secondo di ciascun microservizio	Soddisfatto

Codice	Descrizione	Stato
R-192-F-Ob	Lo Scheduler deve avviare la sincronizzazione della soglia minima di allerta per una merce quando aggiornata	Soddisfatto

Tabella 10: Stato dei requisiti funzionali

4.2 - Grafici riassuntivi

Tutti i grafici qui presenti si riferiscono ai **requisiti funzionali**[G].

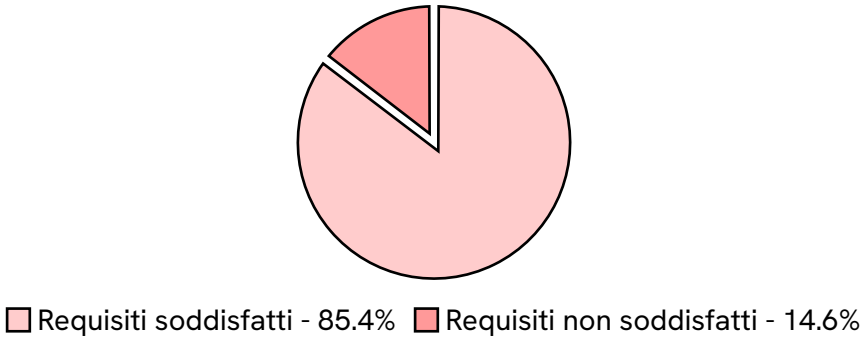

Grafico 152: Percentuale di requisiti funzionali[G] soddisfatti in totale

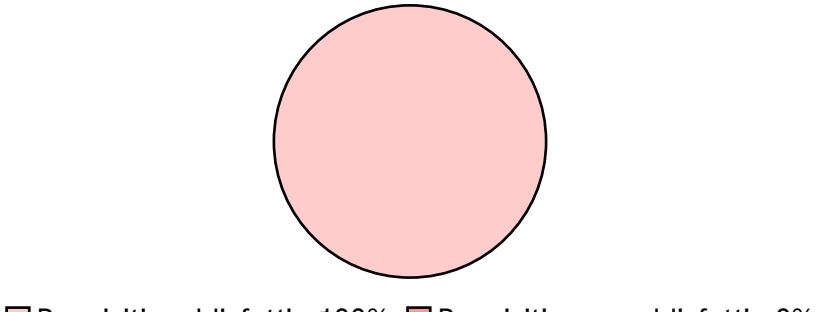

Grafico 153: Percentuale di requisiti funzionali[G] obbligatori soddisfatti

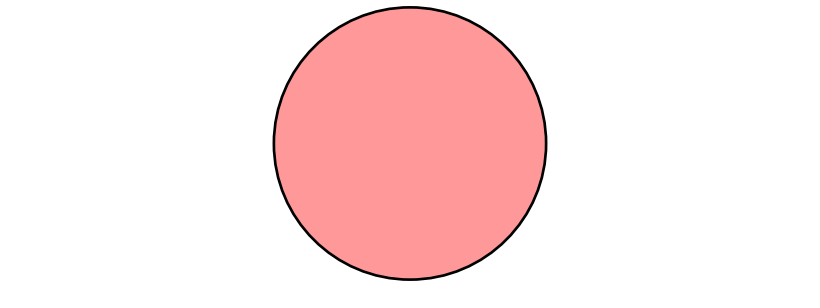

Grafico 154: Percentuale di requisiti funzionali[G] desiderabili soddisfatti

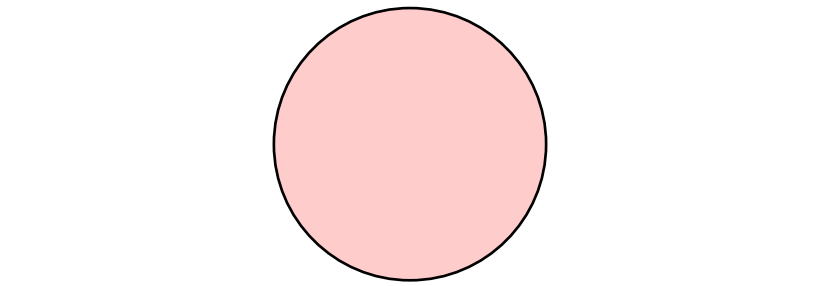

Grafico 155: Percentuale di requisiti funzionali[G] opzionali soddisfatti