Presentazione Architettura

Gruppo n° 8 - A.A. 2024/2025

ALimitedGroup

Limited group, always committed to work

ALimitedGroup

09/04/2025

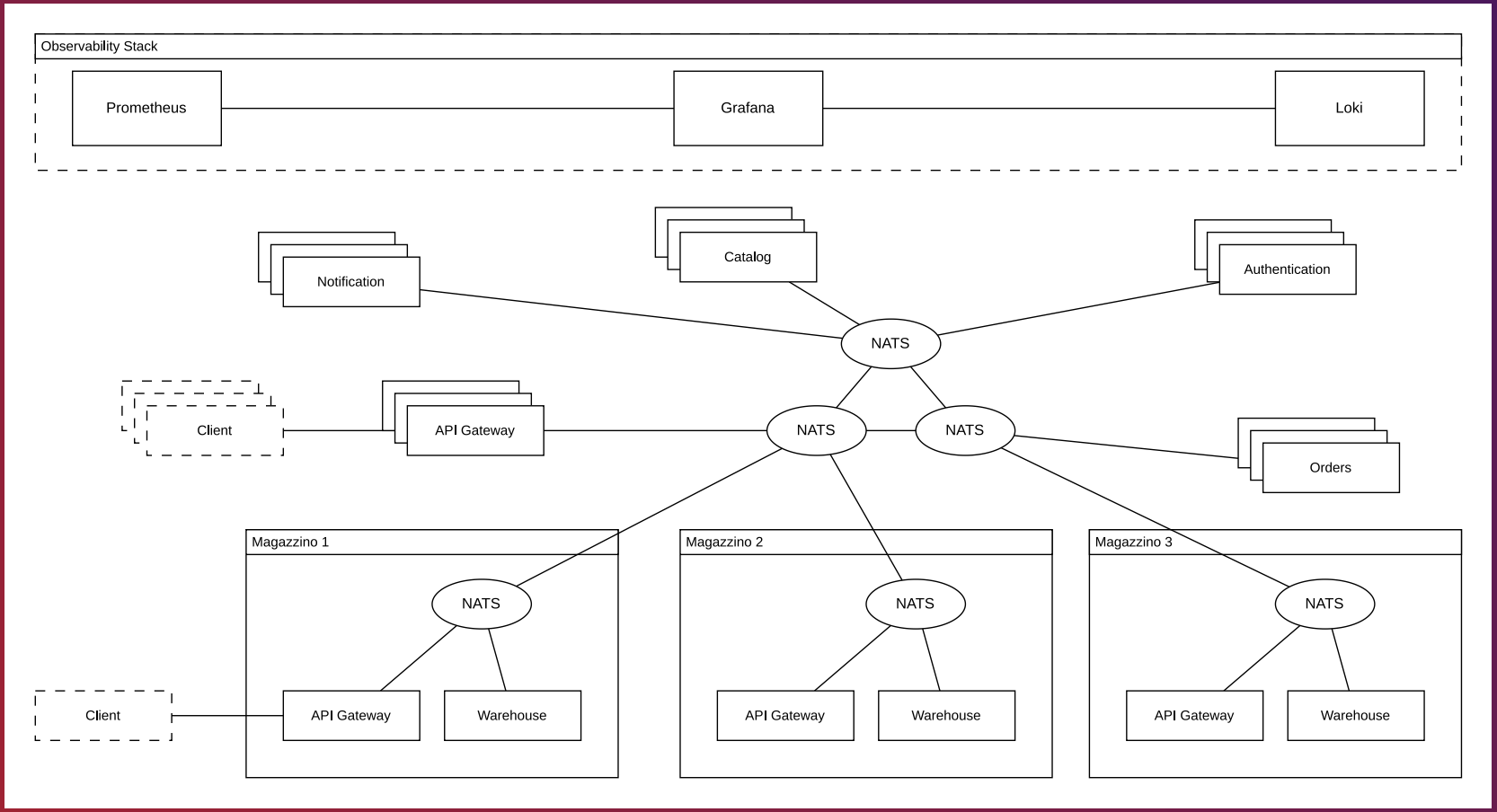
Ripasso capitolato

- Capitolato C6: Sistema di Gestione di un Magazzino Distribuito
- Garantire interoperabilità tra magazzini: ordini, ordini tra magazzini (trasferimenti), notificare se stock scende di soglia
- Accesso al Sistema mediante chiamate API
- Operazione ammissibile solo con token valido e ruolo scelto corretto

Architettura di deployment: Microservizi

- L’idea iniziale del capitolato spinge molto verso questa opzione;
- Maggiore affidabilità;
- Migliore lavoro asincrono, con decomposizione per sottodominio;
- Più facile fare *zero downtime deployment*;
- Possibilità di distribuzione geografica;

Microservizi sviluppati



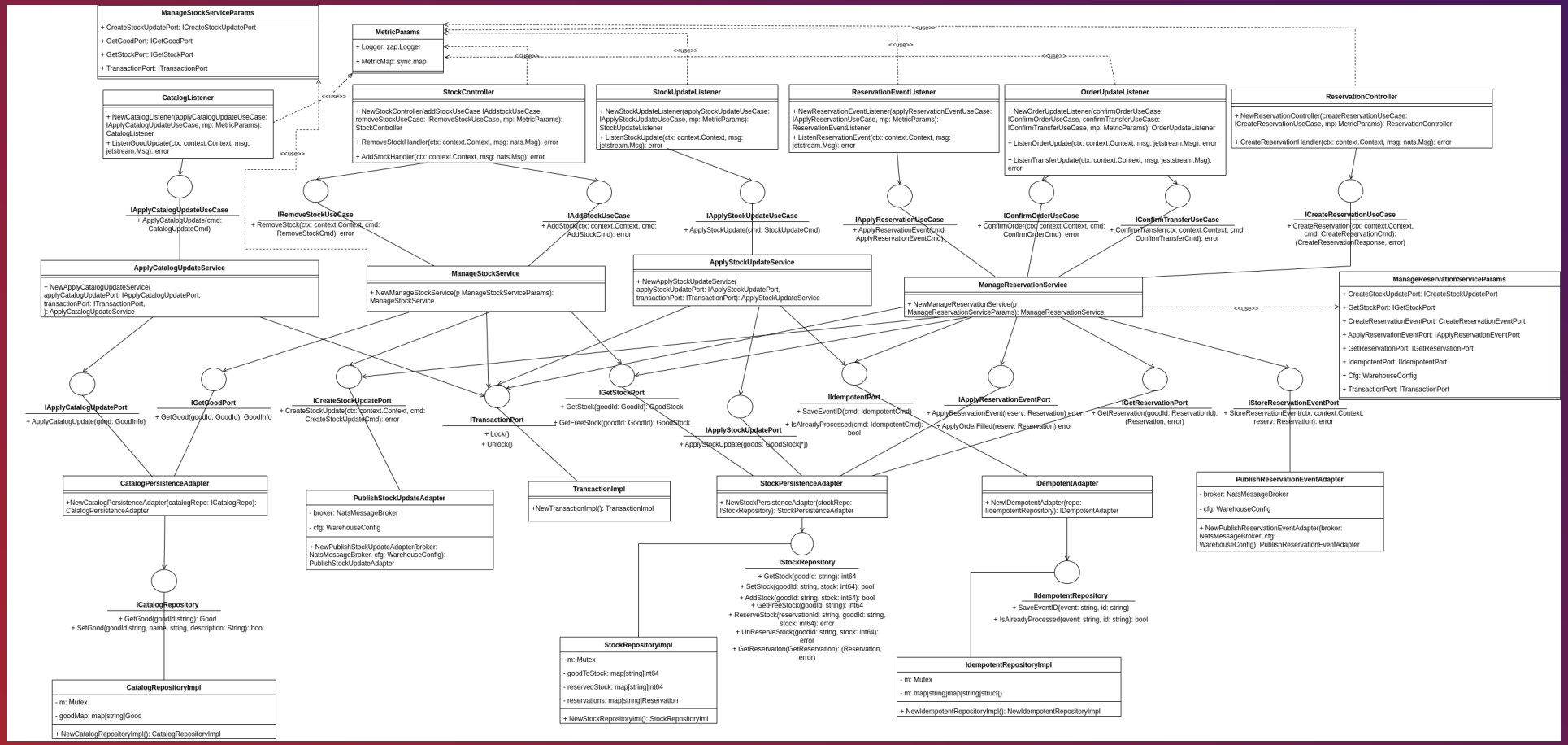
Architettura Esagonale (1/2)

- Abbiamo scelto il pattern dell'Architettura Esagonale per isolare la business logic dalle componenti esterne.
- La comunicazione tra classi avviene tramite interfacce, iniettate nei costruttori, seguendo il principio della Dependency Injection.
- Questo approccio rende il sistema facilmente testabile: la business logic può essere verificata in modo indipendente.

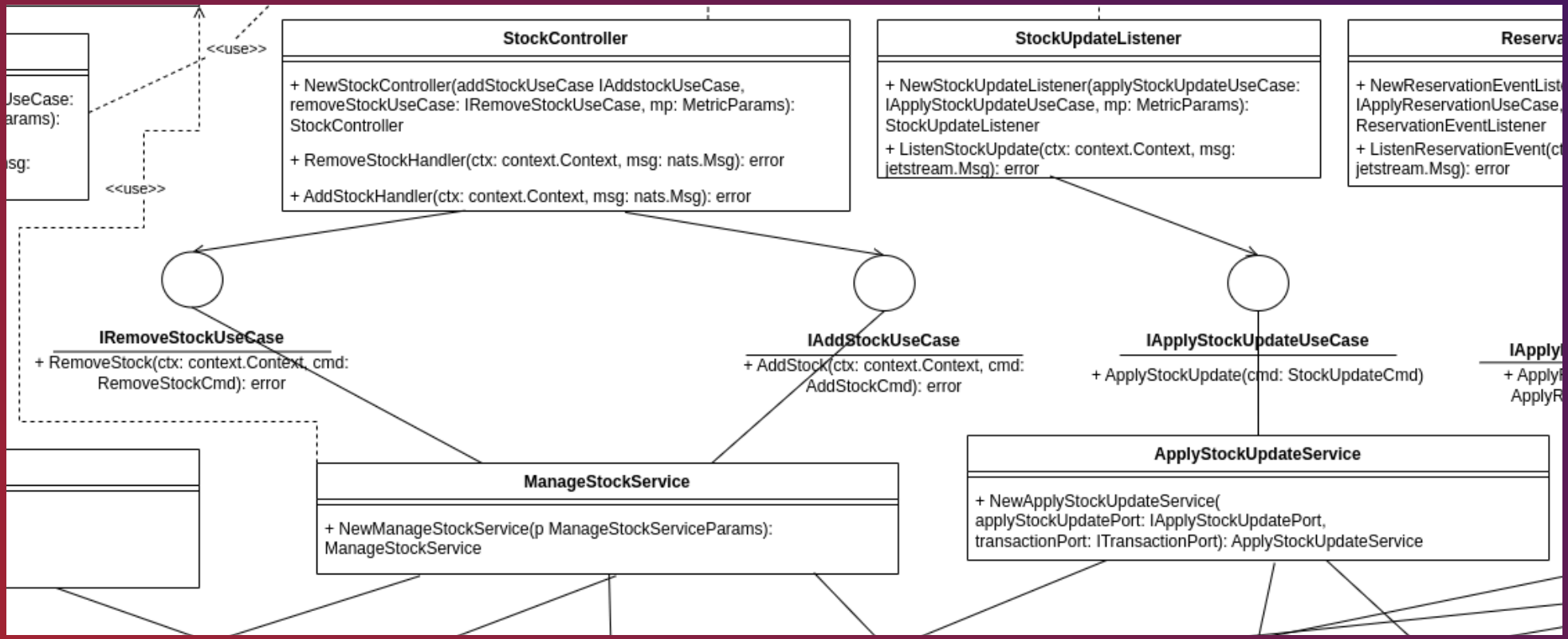
Architettura Esagonale (2/2)

- Le porte d’ingresso e le porte d’uscita definiscono i confini della business logic.
- Questa architettura logica ci permette di cambiare tecnologie senza stravolgere la business logic del servizio
- In un contesto a microservizi, l’architettura esagonale aumenta il disaccoppiamento dei singoli servizi, facilitando l’evoluzione del sistema e la sostituzione di componenti tecnologiche.

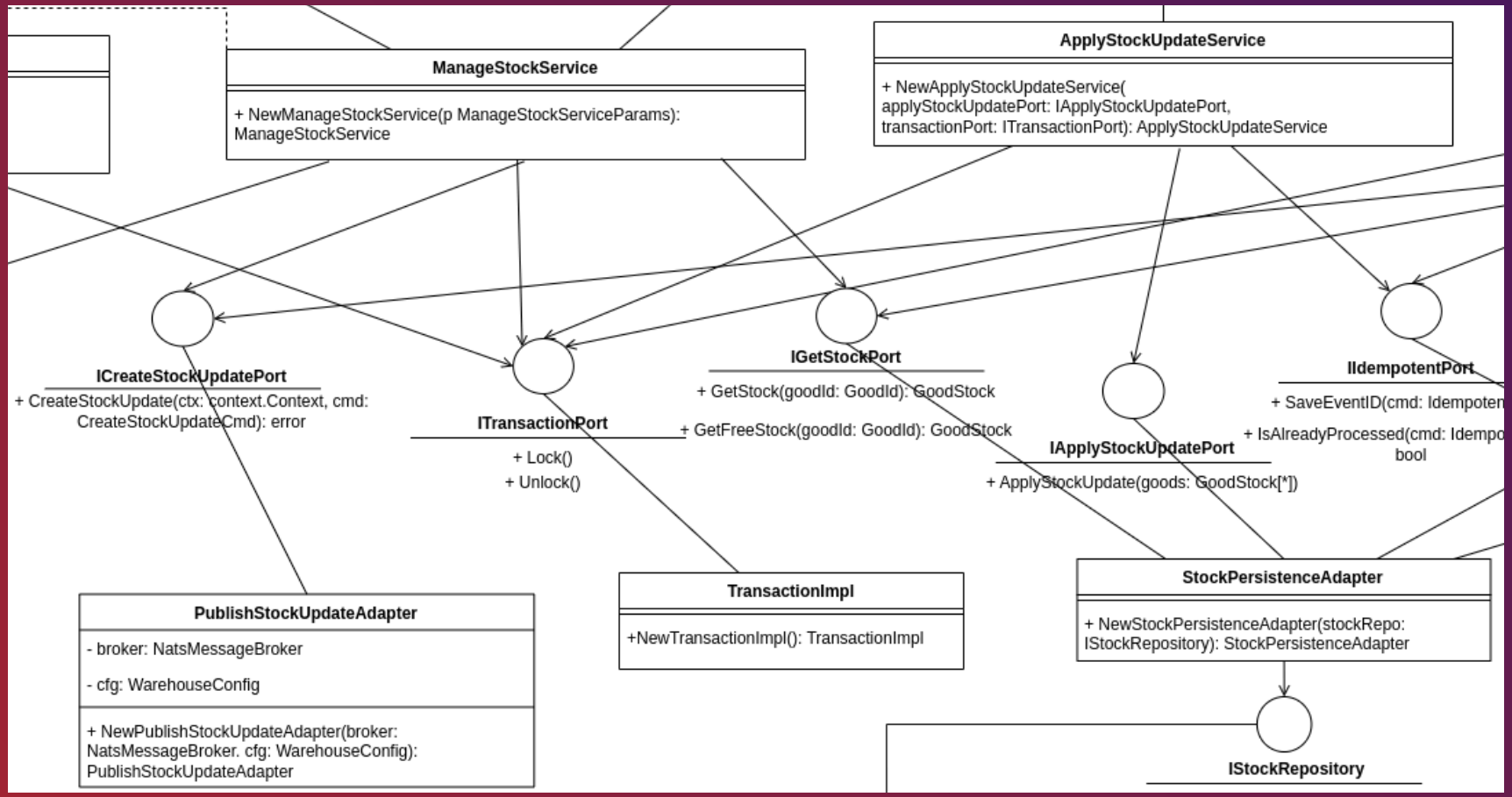
UML Warehouse Microservice (1/4)



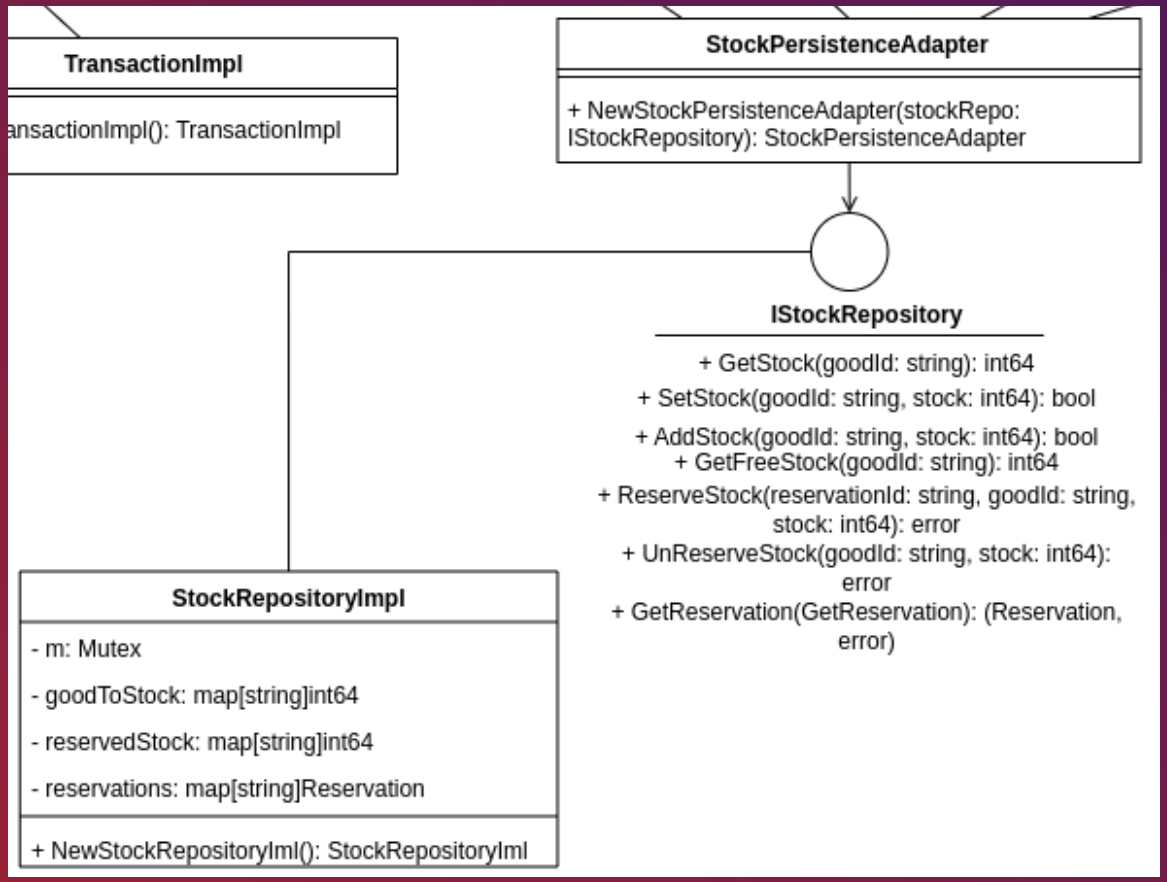
UML Warehouse Microservice (2/4)



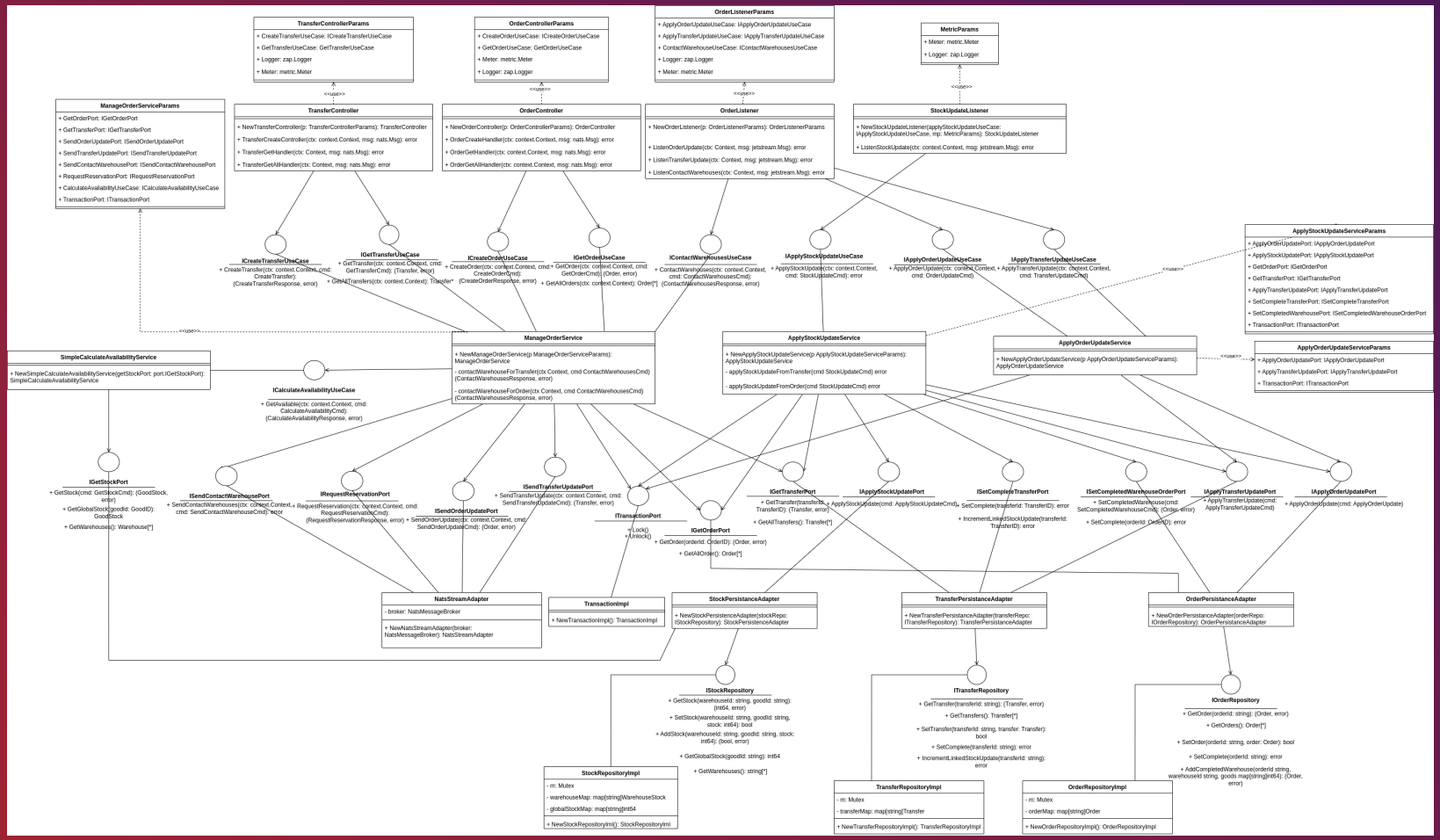
UML Warehouse Microservice (3/4)



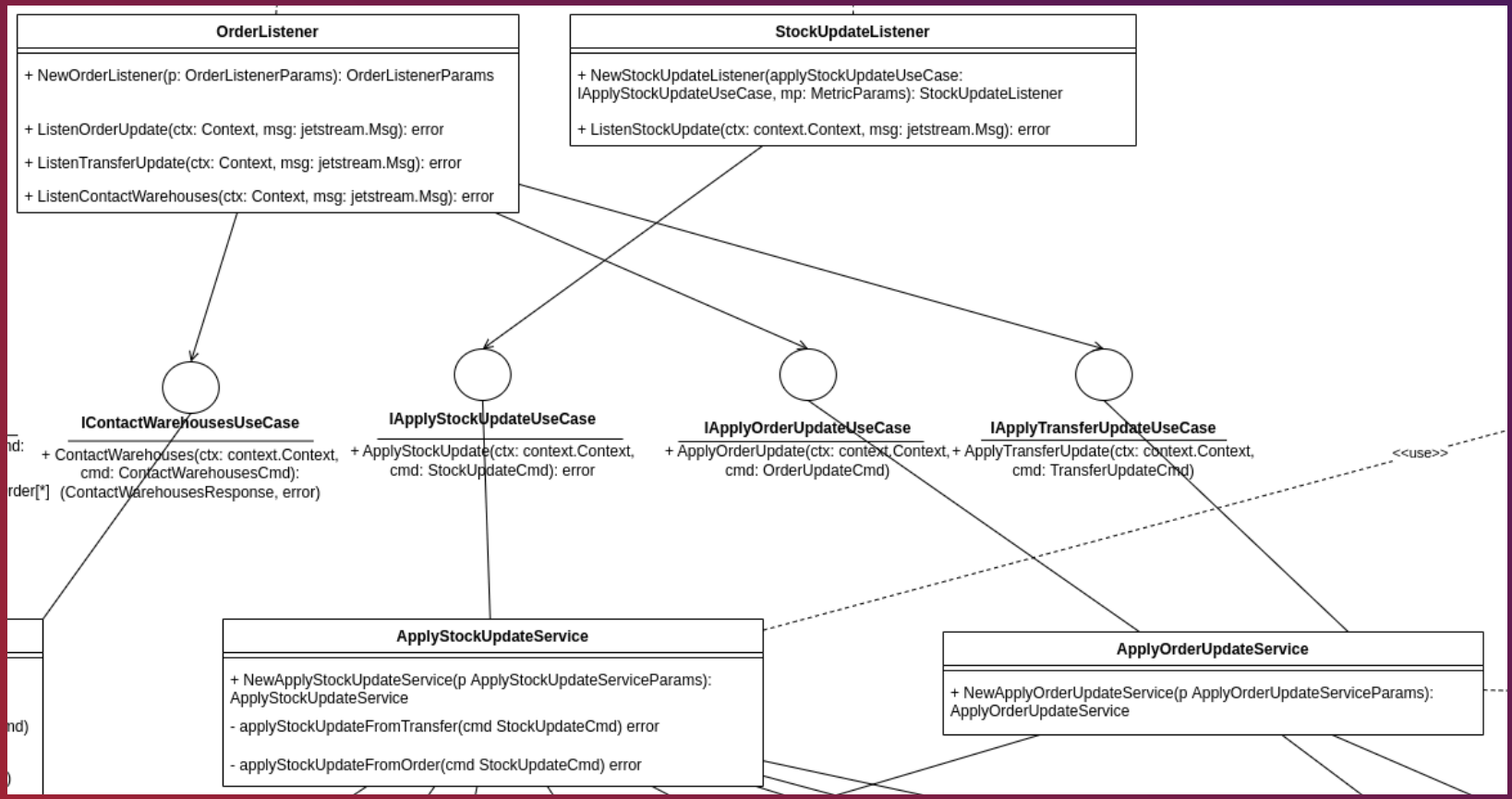
UML Warehouse Microservice (4/4)



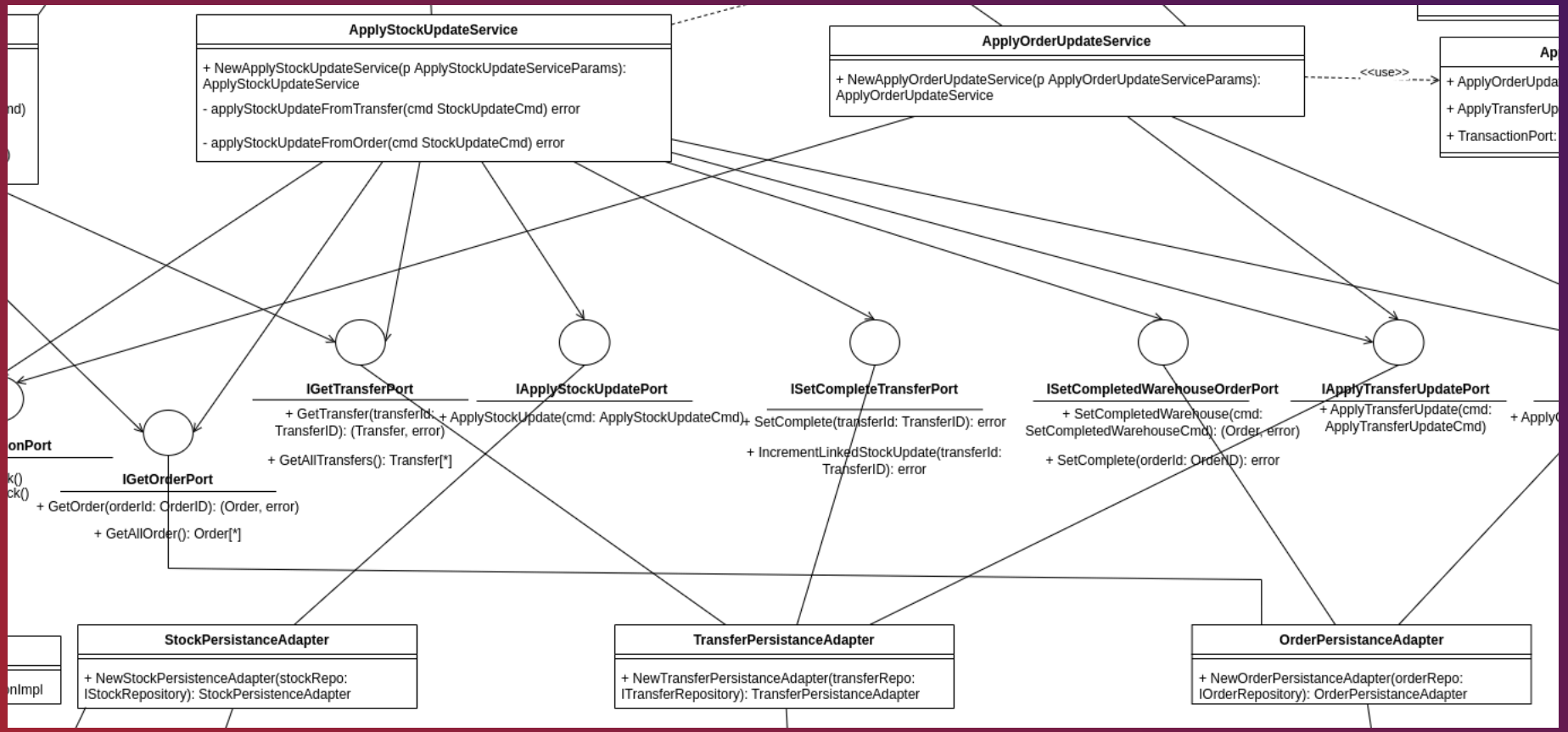
UML Order Microservice (1/4)



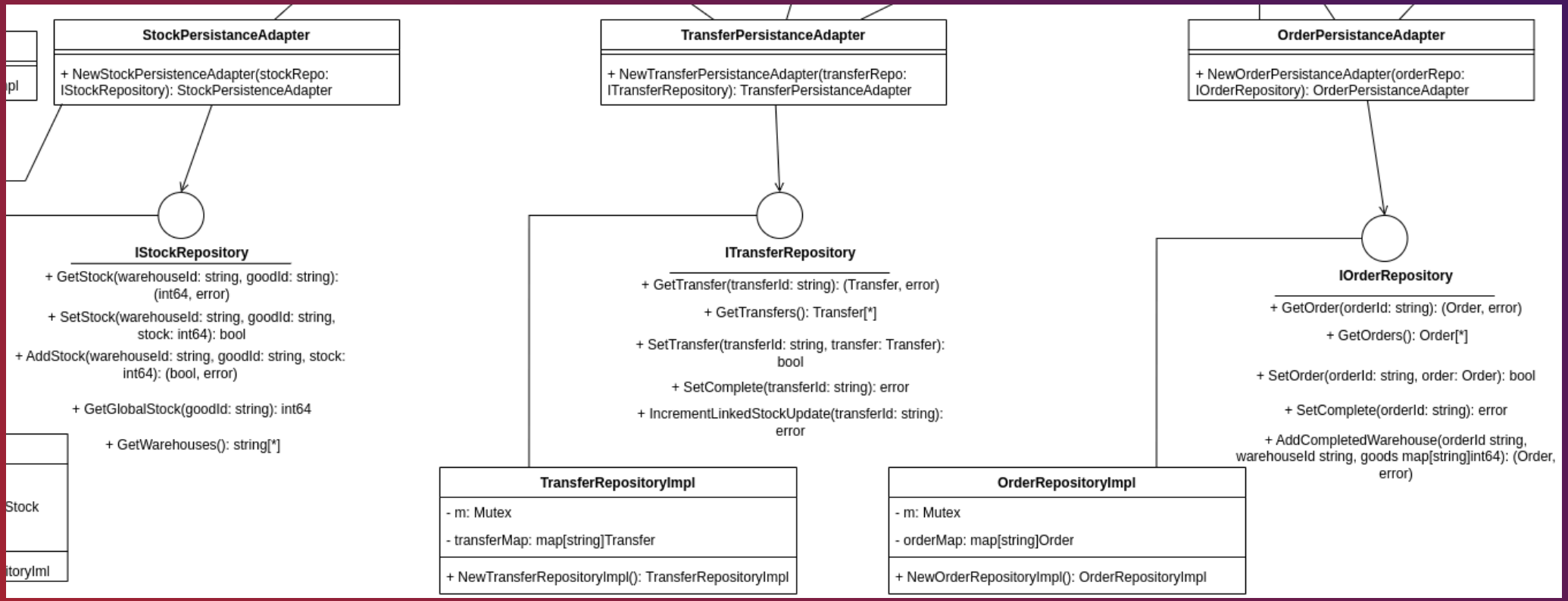
UML Order Microservice (2/4)



UML Order Microservice (3/4)



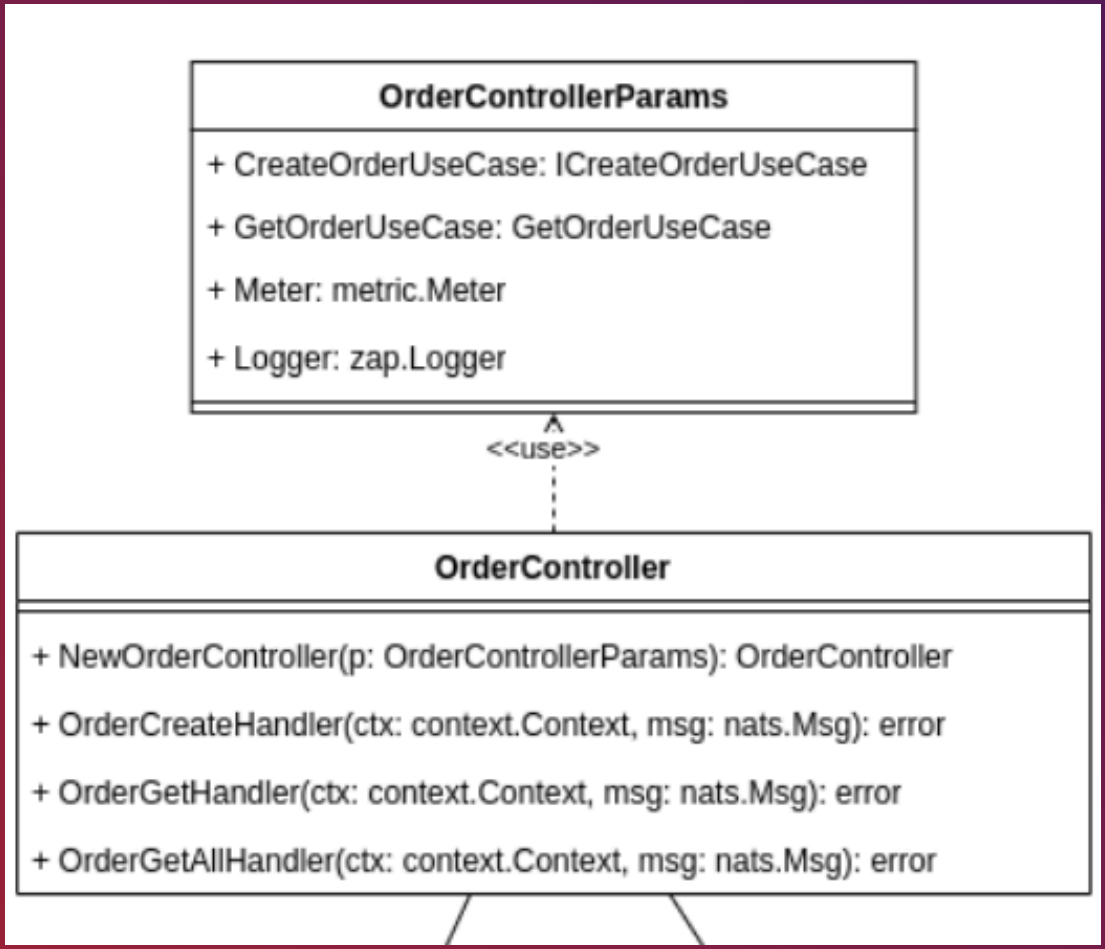
UML Order Microservice (4/4)



Pattern - Dependency Injection

- Molti componenti del Sistema possiedono dipendenze verso altri componenti (esempio, hanno un'istanza di una struttura come attributo)
- Per renderle note e non nascoste abbiamo reso note queste dipendenze inserendole come parametri obbligatori nelle funzioni costruttrici
- Abbiamo utilizzato il framework `fx` di Uber per gestire le dipendenze
- L'utilizzo di un framework ci ha aiutato nella realizzazione dei test e ha semplificato il modo in cui le istanze necessarie vengono fornite

Dependency Injection - Esempio

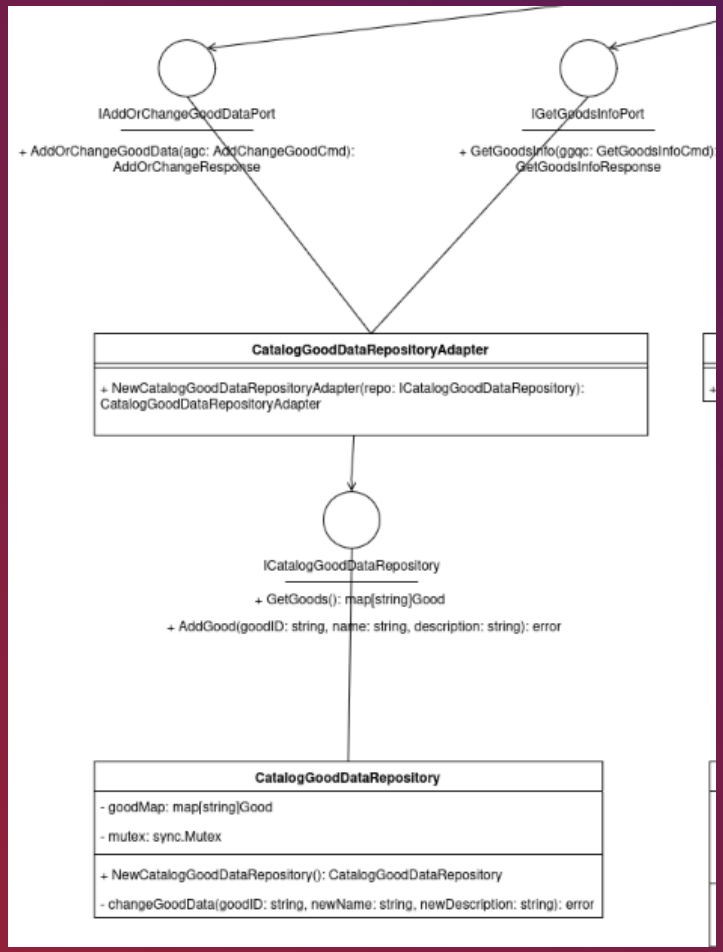


Pattern - Object Adapter

- L’utilizzo dell’architettura esagonale ha reso necessario l’utilizzo di Adapter specie per mettere in collegamento la business logic con la persistence logic

- Gli adapter utilizzati in questo modo implementano delle interfacce della business logic e possiedono un attributo della struttura che ha funzioni della persistence logic

Adapter - Esempio



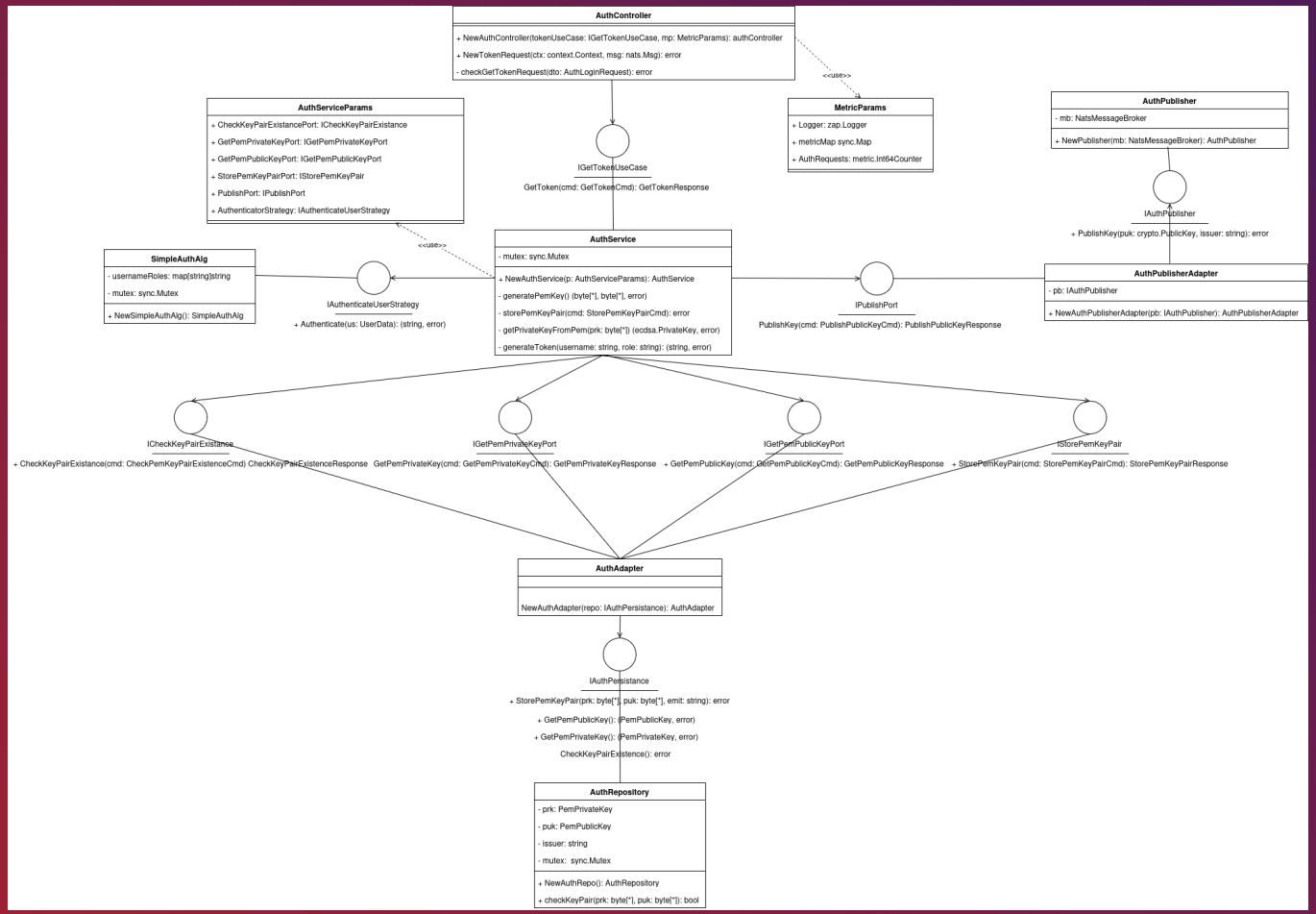
Pattern - Strategy (1/2)

- Attualmente il Sistema non possiede un’autenticazione con nome utente e password, ma fornisce un token di accesso se il ruolo richiesto è valido
- L’azienda proponente, **M31**, ha però richiesto di fare attenzione a questo aspetto in quanto l’autenticazione deve essere facilmente migliorabile in futuro

Pattern - Strategy (2/2)

- Abbiamo dunque realizzato un'interfaccia IauthenticateUserStrategy che fa implementare un metodo che, dati i dati di accesso forniti, ne valuti la correttezza
- È stata una scelta tra questo pattern e il Template Method: il Template Method è tuttavia utile solo se tutti gli algoritmi di autenticazione hanno gli stessi passaggi ma differiscono per il comportamento di almeno uno di essi
- Nel nostro caso, questo può non essere vero specie aggiungendo "livelli" di sicurezza

Strategy - Esempio (2/2)



Strategy - Esempio (2/2)