# Architettura dei sistemi software

## Progetto per l'A.A. 2023/2024

12 dicembre 2023

## Premessa

Il progetto del corso di Architettura dei sistemi software è relativo alla sperimentazione pratica delle tecnologie studiate durante il corso, e riguarda la realizzazione di una semplice applicazione a microservizi ed il rilascio di questa applicazione in un ambiente di esecuzione a container.

Il progetto è basato sullo svolgimento di diverse attività (alcune delle quali sono obbligatorie mentre altre sono opzionali) e con alcune varianti, come descritto nelle seguenti sezioni.

## Progetto (punto di partenza)

**OrderManager** è una semplice applicazione a microservizi per la gestione di ordini di prodotti.

Il codice sorgente di **OrderManager** è disponibile sul repository GitHub del corso (https://github.com/aswroma3/asw/tree/master/projects/asw-order-manager).

L'applicazione **OrderManager** consente di:

- creare prodotti e modificare la loro quantità disponibile (ovvero il loro livello di inventario);
- creare ordini relativi a uno o più prodotti;
- richiedere la validazione di un ordine.

Un ordine si intende valido se l'ordine esiste, se tutti i prodotti ordinati esistono e se inoltre le quantità ordinate di ciascuno dei prodotti ordinati non sono superiori alle quantità disponibili.

(Si tratta di una regola di validazione degli ordini molto semplificata, ma si assuma che questo sia proprio quanto richiesto nel dominio dell'applicazione.)

L'applicazione **OrderManager** è composta dai seguenti microservizi:

- Il servizio **product-service** gestisce i prodotti.
    - Ogni prodotto ha un nome (che lo identifica), la categoria, la quantità disponibile e il prezzo unitario.
    - Un esempio di prodotto:
        - name: *Guerra e Pace*
        - category: *Libro*
        - stockLevel: *3*
        - price: *19.99*
    - Operazioni sui prodotti:
        - POST /products crea un nuovo prodotto (dati nome, categoria, quantità disponibile e prezzo unitario, passati nel corpo della richiesta)
        - GET /products/{name} trova un prodotto, dato il nome
        - GET /products trova tutti i prodotti
        - POST /findproducts/bynames trova tutti prodotti che hanno il nome compreso in una lista di nomi (la lista di nomi è passata nel corpo della richiesta)
        - PATCH /products aggiorna la quantità disponibile di un prodotto (dati nome e variazione della quantità, passati nel corpo della richiesta)

- Il servizio **order-service** gestisce gli ordini.
    - Ogni ordine ha un id (che lo identifica), il cliente, l'indirizzo, un insieme di righe di ordine (ognuna con nome del prodotto e quantità) e il totale.
    - Un esempio di ordine:
        - id: *2*
        - customer: *Woody*
        - address: *Roma*
        - orderItems:
            - product: *Guerra e Pace*, quantity: *2*
            - product: *Anna Karenina*, quantity: *1*
        - total: *50.97*
    - Operazioni sugli ordini:
        - POST /orders crea un nuovo ordine (dati cliente, indirizzo, articoli ordinati e totale, passati nel corpo della richiesta)
        - GET /orders/{id} trova un ordine (dato l'id)
        - GET /orders trova tutti gli ordini
        - GET /findorders/customer/{customer} trova tutti gli ordini di un cliente (dato il cliente)
        - GET /findorders/product/{product} trova tutti gli ordini contenenti un certo prodotto (dato il prodotto)
- Il servizio **order-validation-service** consente di validare un ordine.
    - La convalida di un ordine (l'esito di una validazione) è composta dall'id dell'ordine, alcuni dati dell'ordine (cliente, prodotti ordinati), un indicatore di validità e una motivazione.
    - Un esempio di convalida:
        - id: *6*
        - customer: *Woody*
        - orderItems:
            - product: *Pace e Guerra*, quantity: *1*
            - product: *Anna Karenina*, quantity: *10*
        - valid: *false*
        - motivation: *Il prodotto Pace e Guerra non esiste. Il prodotto Anna Karenina non è disponibile nella quantità richiesta.*
    - Operazioni:
        - GET /ordervalidations/{id} calcola e restituisce la convalida di un ordine (dato l'id)
- Il servizio **api-gateway** (esposto sulla porta 8080) è l'API gateway dell'applicazione, che:
    - espone il servizio **product-service** sul path /productservice; ad esempio, GET /productservice/products
    - espone il servizio **order-service** sul path /orderservice;
      ad esempio, GET /orderservice/orders/{id}
    - espone il servizio **order-validation-service** sul path /ordervalidationservice;
      ad esempio, GET /ordervalidationservice/ordervalidations/{id}

Si veda il repository GitHub del corso per una descrizione di come costruire ed eseguire questa applicazione.

## Discussione

Sul repository GitHub del corso, l'implementazione dell'operazione GET /ordervalidations/ID del servizio **order-validation-service**, per validare l'ordine di identificatore ID, è basata su invocazioni remote REST ai servizi **order-service** e **product-service**, come segue:

- prima viene invocata l'operazione GET /orders/ID di **order-service** per verificare se l'ordine esiste e per trovare i dettagli dell'ordine, e soprattutto l'insieme RR delle righe di ordine dell'ordine;
- poi, per ogni riga di ordine RO di RR, relativa al prodotto P ed alla quantità Q, viene invocata l'operazione GET /products/P di **product-service** per verificare se il prodotto esiste e per trovare i dettagli del prodotto, per verificare che la quantità ordinata Q non sia superiore alla quantità disponibile; (in effetti, l'implementazione fornita per questo punto è leggermente diversa, ma questo non cambia le considerazioni che seguono);
- se tutte le condizioni sono verificate, allora la validità dell'ordine è *true* e la motivazione è *OK*, altrimenti la validità dell'ordine è *false* e la motivazione include una descrizione sintetica di tutte le condizioni che non sono verificate.

Questa implementazione del servizio **order-validation-service** soffre di diverse limitazioni. In particolare, la disponibilità e le prestazioni di questo servizio dipendono fortemente dalla disponibilità e dalle prestazioni dai servizi **order-service** e **product-service**, e se anche uno solo di questi due servizi non è disponibile oppure presenta delle latenze significative allora lo stesso è vero per il servizio **order-validation-service**. Inoltre questo servizio non è molto scalabile, perché nessuno dei servizi è replicato e perché i diversi servizi comunicano in modo sincrono.

L'operazione di validazione di un ordine è un'operazione importante per l'applicazione **OrderManager**, e pertanto va implementata in modo da sostenere prestazioni, disponibilità e scalabilità.

## Progetto (attività)

Il progetto consiste nel modificare l'applicazione presente sul repository GitHub del corso, svolgendo una o più tra le seguenti attività (alcune delle quali sono obbligatorie me altre sono opzionali).

### *Modifiche del codice e della configurazione dell'applicazione (obbligatorio)*

Una prima modifica (obbligatoria) riguarda la modifica del codice e della configurazione dell'applicazione, come segue:

- Nei servizi **product-service** e **order-service** bisogna usare una base di dati MySQL o PostgreSQL al posto di HSQLDB (le due basi di dati vanno eseguite in due container Docker separati).

Una seconda modifica (obbligatoria) riguarda la modifica del codice e della configurazione dell'applicazione, come segue:

- Modificare la logica del servizio **order-validation-service**, come descritto nel seguito.

Si potrebbe pensare di modificare la logica del servizio **order-validation-service** utilizzando delle invocazioni remote *asincrone* (anziché *sincrone*), ed eseguire l'algoritmo per la validazione di un ordine nel modo più concorrente possibile. Ma anche in questo modo la disponibilità del servizio **order-validation-service** continua a dipendere fortemente dalla disponibilità dei servizi **product-service** e **order-service**. Dunque, bisogna cercare una soluzione migliore.

Una soluzione probabilmente migliore consiste invece nell'invertire la logica del servizio **order-validation-service**, come segue:

- Quando viene aggiunto un nuovo ordine, il servizio **order-service** deve notificare un evento **OrderCreatedEvent** (con tutti i dati importanti dell'ordine), su un apposito canale per messaggi.
- In modo analogo, quando viene aggiunto un nuovo prodotto, il servizio **product-service** deve notificare un evento **ProductCreatedEvent** (con tutti i dati importanti del prodotto), su un apposito canale per messaggi.
- Inoltre, quando viene aggiornata la quantità disponibile di un prodotto, il servizio **product-service** deve notificare un evento **ProductStockLevelUpdatedEvent** (con tutti i dati importanti dell'evento), su un apposito canale per messaggi.
- Il servizio **order-validation-service** deve gestire una propria base di dati (separata dalle precedenti, in un container Docker separato), con una tabella per gli ordini e una tabella per i prodotti, per memorizzare tutti e soli i dati sugli ordini e sui prodotti che sono necessari per effettuare la validazione degli ordini.
- Ogni volta che il servizio **order-validation-service** riceve un evento di tipo **OrderCreatedEvent** allora deve aggiornare la propria tabella degli ordini.
- In modo analogo, ogni volta che il servizio **order-validation-service** riceve un evento di tipo **ProductCreatedEvent** oppure **ProductStockLevelUpdatedEvent** allora deve aggiornare la propria tabella dei prodotti.
- Il servizio **order-validation-service** potrà poi rispondere alle richieste GET /ordervalidations/{id} accedendo solo alla propria base di dati.

I canali per messaggi possono essere gestiti con Apache Kafka (in esecuzione in un container Docker separato) oppure anche un message broker differente.

Nel realizzare queste modifiche, si raccomanda di mantenere l'architettura esagonale dei diversi servizi; in particolare:

- la logica di business va collocata nell'interno di ogni servizio (ovvero nel package **domain**), comprese le porte; nessuna logica infrastrutturale nell'interno di un servizio;
- le responsabilità infrastrutturali (gli adattatori) vanno collocate nell'esterno di ogni servizio (ovvero, in tutti gli altri package); nessuna logica di business negli adattatori.

### *Modifiche della modalità di rilascio dell'applicazione*

Delle ulteriori modifiche riguardano la modifica della modalità di rilascio dell'applicazione, sulla base delle seguenti possibili attività e varianti:

- Eseguire i diversi servizi ciascuno in un proprio container Docker (obbligatorio).
- Mandare in esecuzione più istanze di ciascun servizio (obbligatorio). Per semplicità, le basi di dati non vanno replicate.
- Eseguire i diversi servizi in container Docker, usando Docker Compose (opzionale).
- Eseguire i diversi servizi in container Docker, usando Kubernetes (opzionale).

## Modalità di svolgimento e di consegna del progetto

Il progetto va svolto in gruppi composti preferibilmente da 3-5 studenti.

Ciascun gruppo dovrà interagire con il docente in questo modo:

- Ciascun gruppo dovrà comunicare al docente, per posta elettronica, appena possibile, la composizione del proprio gruppo, la soluzione che si intende implementare e le tecnologie che si intendono utilizzare. Sia la soluzione che le tecnologie potrebbero essere diverse da quelle proposte in questo documento.
- Poi, ciascun gruppo dovrà realizzare e verificare (sul proprio computer) la propria applicazione distribuita.
- Infine, ciascun gruppo dovrà caricare la propria applicazione distribuita su GitHub (o altro servizio di condivisione del codice). In particolare, dovrà caricare tutto il codice sorgente dell'applicazione, oltre a ogni script necessario per la compilazione e costruzione dell'applicazione (Gradle o Maven), i file Dockerfile e gli eventuali file per Docker Compose o Kubernetes, nonché gli script per mandare in esecuzione l'applicazione.
- Al completamento del progetto, il gruppo dovrà comunicare al docente, per posta elettronica, l'URI su GitHub del codice dell'applicazione, insieme a una descrizione sintetica della soluzione effettivamente implementata, delle tecnologie effettivamente utilizzate e delle attività svolte.

## Valutazione del progetto

La valutazione del progetto prenderà in considerazione diversi aspetti, e in particolare:

- Le realizzazioni delle modifiche indicate come "obbligatorie", e la loro correttezza.
- Quante e quali varianti indicate come "opzionali" sono state realizzate, e la loro correttezza.
- Il rispetto dell'architettura esagonale.

## Altri progetti

Ciascun gruppo può formulare, se vuole, una propria proposta di progetto (che deve comunque avere finalità simili a quelle del progetto illustrato in questo documento). In ogni caso, queste proposte di progetti alternativi devono essere autorizzate preventivamente dal docente.