Relazione progetto finale

E-commerce Product Management

Distributed Systems and Big Data

C.d.L.M. Ingegneria informatica

Prof.ssa: **Antonella Di Stefano**

Cristaudo Gianluca – Mat. 1000012305

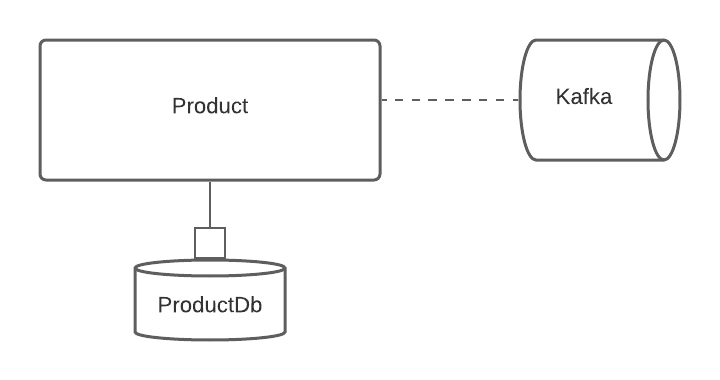
Germano Giuseppe – Mat. 1000008784

Introduzione

Per questo progetto finale, abbiamo scelto di implementare il progetto 1 nella variante “A” il quale prevede la realizzazione di un micro-servizio che si occupa della gestione dei prodotti, scegliendo di:

* Effettuare lo sviluppo con Java Spring MVC
* Utilizzare JPA
* Utilizzare MySql

L’architettura generale che è stata realizzata è la seguente:



La struttura del micro-servizio si divide nelle seguenti sotto-specializzazioni:

* Controller
* Repository
* Service
* METTERE SCREEN

Il pattern cosi composto garantisce una perfetta suddivisione delle responsabilità definendo un Controller/Handler delle richieste da parte di un client che a sua volta inoltrerà la data operazione ad un servizio la cui responsabilità è inglobare la logica di business, a sua volta tale servizio farà uso dei repository per poter interfacciarsi al database.

Attraverso JPA, vengono formalizzate le entità del database e i relativi constraints nonché le relazioni fra queste.

**Logica Controller**

Attraverso la annotation *controller* deleghiamo alla classe Java la responsabilità di “ricevere”, e successivamente inoltrare ai servizi, i comandi REST API che definiscono le CRUD operation.

Attraverso le annotazioni GET/POST Mapping definiamo un URI fittizio, quindi esponendo tale API all’esterno si permette ad un client o ad un ulteriore servizio si permette di eseguire operazioni CRUD passando un parametro all’interno della richiesta, durante lo scambio di dati viene utilizzato il formato JSON specificato nel content-type di HTTP.

All’interno della sotto-cartella jsonModel sono presenti tutte le classi che fungono da modello come formato di scambio dati attraverso JSON.

Sono presenti i seguenti Controller:

* CategoryController: si occupa della gestione delle varie categorie.
* PingController: si occupa di implementare un semplice algoritmo di basic ping.
* ProductController: si occupa della gestione dei vari product.

**API**

* <http://localhost:3333/categories/product> (POST): Richiede l’aggiunta di una serie di categorie ad un determinato prodotto.
* <http://localhost:3333/categories/add> (POST): Richiede l’aggiunta di una categoria.
* [http://localhost:3333/categories/del/{id}](http://localhost:3333/categories/del/%7bid%7d) (DELETE): Richiede l’eliminazione di una categoria specifica.
* <http://localhost:3333/product/add> (POST): Richiede l’aggiunta di un prodotto.
* <http://localhost:3333/product/all> (GET): Richiede l’intera lista dei prodotti presenti.
* <http://localhost:3333/product/all?per_page=10&page=0> (GET): Richiede l’intera lista dei prodotti presenti secondo una formattazione specifica.
* http://localhost:3333/product/id/{id} (GET): Richiede tutti i dati di un prodotto specifico.
* <http://localhost:3333/product/update> (PUT): Richiede l’update di un prodotto.
* <http://localhost:3333/ping> (GET): Richiede lo stato del servizio e del database.

**Logica Repository e Service**

L’annotazione Service definisce le responsabilità di questa classe Java, qui viene inglobata la logica di business nonché l’interfacciamento del repository al database.  
Vengono formalizzate le CRUD operation richieste dall’utente/servizio dopo essere state “filtrate” dal controller.

Sono presenti i seguenti Service/Repository:

* Category
* Product

**Logica Gestione Eccezioni**

La gestione delle eccezioni avviene all’interno di una classe (che estende la classe ResponseEntityExceptionHandler di Spring) facendo uso della annotazione *ControllerAdvice*, all’interno della classe vengono definiti e ridefiniti qualora fossero già presente delle implementazioni di default del framework Spring (Override) i metodi Handler per la gestione delle seguenti eccezioni, che tornano l’http status:

* ProductNotFound: nel caso di prodotto non trovato restituisce NotFound.
* CategoryNotFound: nel caso di categoria non trovata restituisce NotFound.
* NoHandlerFoundException(Override): nel caso di URI errata restituisce Bad Request.
* HttpMessageNotReadable(Override): nel caso di formato JSON errato restituisce Bad Request.
* HttpRequestMethodNotSupported(Override): nel caso di richiesta di un metodo non supportato restituisce Bad Request.

**Kafka**

Al fine di mettere in comunicazione microservizi distinti, in questo caso ordini e prodotti, si è fatto uso del broker Kafka.  
Il Microservizio in tal caso implementa 2 classi:

* Consumatore: sul topic *orders* con lo scopo di validare i prodotti,quantità e prezzo al fine di approvare l’ordine.
* Produttore: una volta che il messaggio viene validato viene pubblicato sul topic *orders*, evitando che questo stesso lo consumi nel suo ruolo di produttore.  
  Si comporta da produttore per altri topic (Logging, Notifications) in caso di errore di validazione dell’ordine.  
    
  Segue una lista con i principali comandi utilizzati:

Il primo comando accede al servizio kafka.  
Il secondo comando avvia un produttore sul topic orders specificando chiave-valore e separatore.  
Il terzo comando avvia un consumatore sul topic orders specificando chiave-valore e separatore.  
Il quarto comando stampa una lista di tutti i topic.  
Il quinto comando simula un esempio di messaggio contente chiave-valore separati da una “,” (ordine completato).

**Entities**

Attraverso JPA vengono create le entità necessarie alla gestione dei dati relativi all’applicazione facendo uso delle seguenti classi java:

* Category
* Product
* Product\_categories

Al fine di ottenere le relazioni desiderate fra categorie e prodotti (ovvero che un prodotto appartiene a più categorie e viceversa) si è fatto uso della seguente strategia, nelle classi Categoria e Prodotti è stata inserita un attributo di tipo lista che formalizza l’associazione n-n.

Nei parametri dell’annotazione ManyToMany abbiamo definito il Cascade in modo tale da modificare il comportamento di default a seguito di un delete.   
Il comportamento di default (ALL,REMOVE) a seguito di una cancellazione di una categoria/prodotto comporta la cancellazione, non solo del record della tabella intermedia, ma di tutto il grado delle relazioni che da qui in poi si diramano.

L’annotazione JoinTable ci permette di definire manualmente i vincoli di chiavi esterne (foreign key) permettendo la cancellazione che di default fallirebbe a causa del vincolo di chiavi esterne, di conseguenza modificando esplicitamente tali parametri è possibile proseguire con un’operazione di remove.  
Attraverso l’attributo name dell’annotazione JoinTable esposto unico in entrambe le classi (Category e Product) ci permette di definire in modo non ambiguo il nome da assegnare all’unica tabella intermedia che formalizza la relazione.  
Omettendo l’annotazione JoinTable e il parametro di tipo name si nota come il database tratti le relazioni n-n senza preoccuparsi di normalizzare le tabelle (in questo caso in terza forma normale).

Un ulteriore annotazione utile è stata JsonIgnoreProperties che permette di evitare che una volta richiesto un prodotto/categoria, e mostrate le sue specifiche fra cui anche la lista di categorie/prodotti, non mostri ulteriormente la lista delle categorie/prodotti finendo inevitabilmente in una iterazione infinita.

Diagramma UML delle classi:

