

**INF3710- Bases de données**

Groupe 02

**Travail pratique #5**

**Interface SQL et persistance**

**Présenté à Manel Grichi**

**David TREMBLAY 1748125**

**Dominique PICHÉ 1766981**

Département de génie informatique et génie logiciel

Le 13 novembre 2017

École polytechnique de Montréal

# Question 1

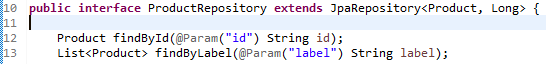
**Analyse du code**

**Lignes 11 et 13 de la classe CustomerRepository**



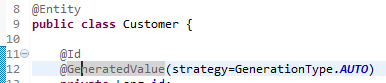
On remarque que CustomerRepository hérite de JpaRepository ainsi que de PagingAndSortingRepository. JpaRepository fait référence au Repository du Java Persistence API (JPA). CustomerRepository est simplement un interface pour manipuler le Repository des Customers. À la ligne 11, on remarque également la notation <Customer, Long>, où Customer réfère au type d’entité, et Long est le type de variable utilisé pour le id du Customer. À la ligne 13, on définit une fonction de recherche, qui retourne une liste des Customer avec un nom de famille spécifique. @Param est une annotation.

**Lignes 10, 12 et 13 de ProductRepository**



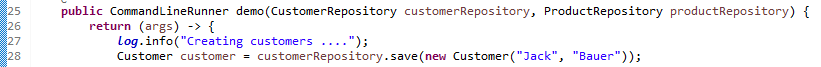
ProductRepository hérite de JpaRepository. C’est une interface pour manipuler le Repository de Product. Il y a ici deux méthodes de recherche, une qui trouve un Product par son id, un autre qui trouve une liste de Product qui ont un label spécifique.

**Lignes 8, 11 et 12 de la classe Customer**



La ligne 8 est une annotation qui spécifie que la classe suivante représente un type d’entité pour laquelle on aura un repository. La ligne 11 est une annotation qui signifie que la prochaine ligne de code (soit le private Long id à la ligne 13) sera utilisée comme primary key pour l’entité. À la ligne 12, on a une annotation qui stipule que le primary key, soit le private Long id, sera généré automatiquement.

**Lignes 25, 28, 67 et 68 de la classe Application**

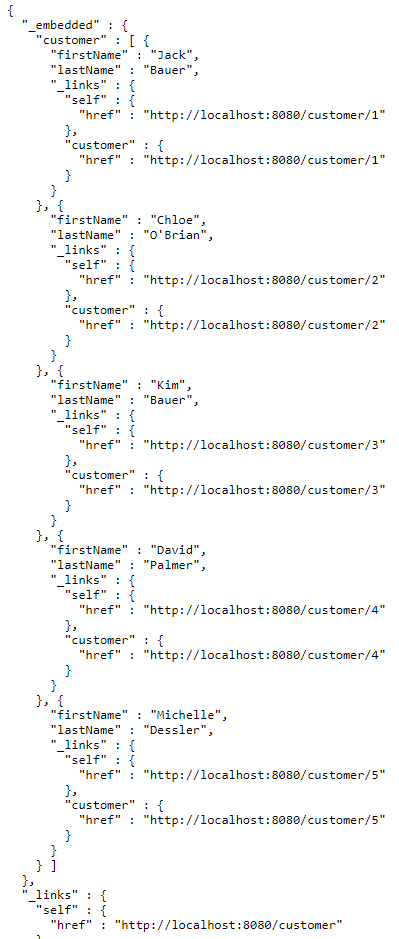


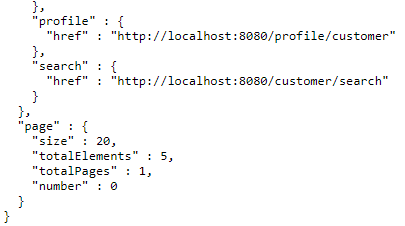
À la ligne 25, on a la définition d’une méthode de démonstration, qui prend en paramètre nos deux repository. Cette méthode effectue, à la ligne 28, une nouvelle entité Customer nommée Jack Bauer et effectue sa sauvegarde dans le customerRepository. Le id de Jack Bauer sera généré automatiquement. À la ligne 67, un nouveau produit est créé, avec le label « How to Use JPA » et la description « Book about JPA advancedfeatures ». Cette nouvelle entité est sauvegardée dans le productRepository. À la ligne 68, un produit au niveau application est créé à partir d’une requête dans le productRepository qui retournera le produit sauvegardé à la ligne précédente (car on passe en paramètre le id de ce produit à l’opération d’entité findById).

**http://localhost:8080/product** :



**http://localhost:8080/customer :**

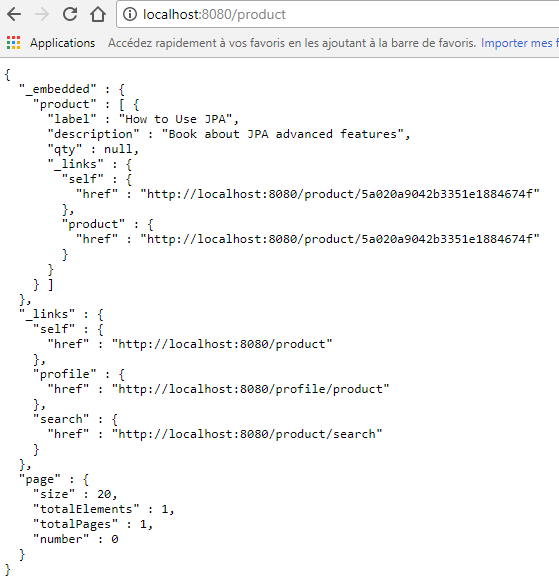




Il s’agit d’une base de données relationnelle, car les entités doivent suivre une structure spécifique, soit celle décrite dans Customer.java ou bien dans Product.java. De plus, ces entités ont des clés bien définies. Les données sont celles qu’on a créées dans la méthode CommandLineRunnerdemo dans Application.java

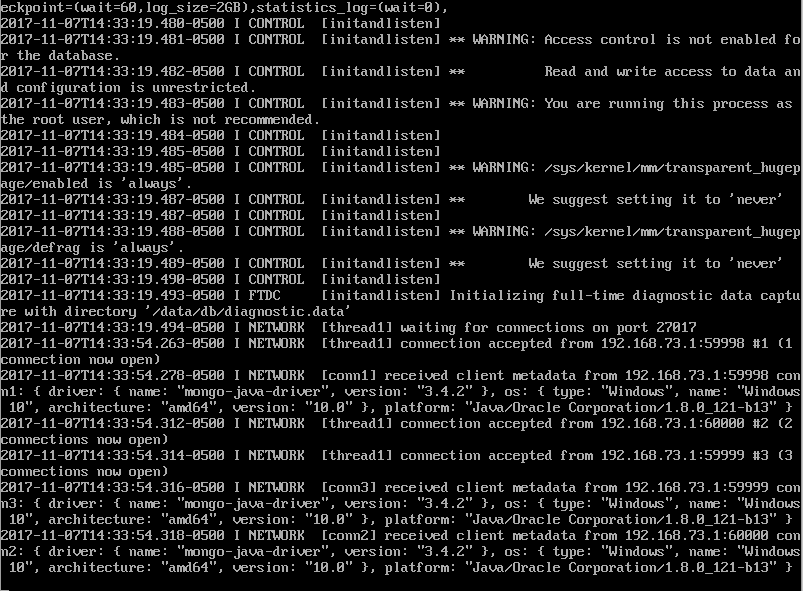
# Question 2

Voici le résultat sur **http://localhost:8080/product** après les modifications :



La seule différence visible est que la clé est beaucoup plus complexe ici. Au lieu d’être « 1 », elle est un chiffre hexadécimal d’une longueur de 24 caractères.

**Shell Mongo :**

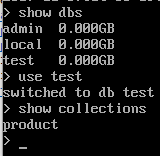


Le shell mongo a accepté 3 connections provenant de l’IP 192.168.73.1 (soit l’adresse IP locale de Windows) sur les ports 59998, 59999 et 60000 et du metadata pour chaque connexion.

La base de données est maintenant non-relationnelle (car on utilise mongo). Cependant, le client y accédant à travers l’application java n’y accédera que comme si c’était une base de données relationnelle, car l’application le force à suivre son schéma.

L’impact est que mongo se charge de la création de la base de données associée au repository, mais l’interface reste sensiblement la même du côté client. Preuve à l’appui : on a simplement changé le repository duquel hérite ProductRepository. Le reste du code reste identique.

Le principal avantage de mongo est qu’on peut ouvrir un client mongo, et venir directement modifier la base de données. On peut, par exemple, modifier les champs d’un produit s’y trouvant déjà avec les requêtes mongo (comme nous le ferons à la prochaine question). On ne souffre pas des mêmes restrictions que lors de l’utilisation d’une base de données relationnelle, c’est-à-dire qu’on peut venir, par exemple, rajouter des champs à une entité spécifique, sans devoir le rajouter aux autres entités du même type.



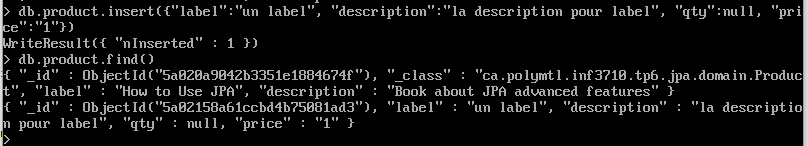
# Question 3



Le prix n’apparaît pas sur **localhost:8080/product :**

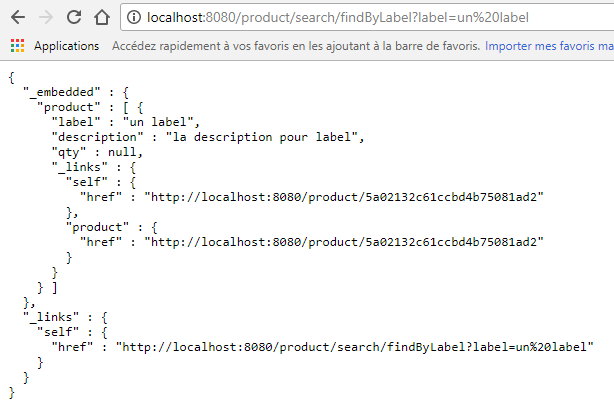


Cependant, il apparaît bien lorsqu’on fait find dans le client mongo :



On remarque aussi que le produit initialement inséré par l’application java a un champ « \_class », qui n’apparaît que dans le find mongo.

**Résultat du findByLabel** :



**Résultat du findByLastName :**



Après modification du code, le prix apparaît sur **le localhost :8080/product.**



Nous avons ajouté les getters et setters au projet pour le nouvel attribut Price. Ils peuvent donc maintenant être affichés sur l'adresse du localhost et non seulement dans le shell Mongo.