Gestion des avis (partie 02)

# TP02 du module 03 – Spring Data MongoDB

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Création d’un Document avec une clef composite * Création du Repository associé * Utilisation de tests unitaires pour valider |

|  |
| --- |
| **Durée estimée** |
| 45 minutes |

Contexte

En vous appuyant sur l’ensemble des concepts abordés dans le cours, mettez en place une application Spring Boot permettant la gestion des avis d’une cave à vin.

L’application de cave à vin sera une application d’API Web sécurisée à la fin des itérations.

L’application de gestion des avis sera aussi une application d’API Web :

* Elle doit permettre de gérer les avis des clients sur le vin commandé.
  + La date de l’avis
  + L’identifiant de la bouteille commandée, ainsi que de la Région et couleur
  + La quantité commandée par le client
  + Une note (entre 0 et 5), un commentaire et le pseudo du client (permettre de protéger les informations du client RGPD)
* Le but étant de faire des statistiques pour le propriétaire du Vignoble

Dans les itérations de ce module ; nous réaliserons uniquement du back de l’application. En particulier, la couche DAL et les BO pour le moment.

Cette application se fera avec Spring Boot.

Déclarer la structure des données en Anglais

* Voici le diagramme de classes pour cette itération

Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

* Voici le diagramme de la Collection en base :

Une image contenant texte, capture d’écran, Police, algèbre

Description générée automatiquement

Solution

# Créer le Document Bouteille et sa classe de clef composite

* Le package des classes est fr.eni.gestionavis.bo.vin
* Utiliser les annotations de Spring Data MongoDB
* La classe de la clef composite doit respecter le design pattern POJO
  + Elle s’appelle BouteilleId
  + Elle a 3 attributs qui composent la clef
    - idBouteille
    - idRegion
    - idCouleur
  + Ajouter les annotations Lombok : @Data, @AllArgsConstructor, @NoArgsConstructor et @Builder
    - Se rappeler qu’elle doit bien redéfinir la méthode equals
    - Ce qui est le cas avec @Data
  + Elle doit implémenter Serializable

**package** fr.eni.gestionavis.bo.vin;

**import** java.io.Serializable;

**import** org.springframework.data.mongodb.core.mapping.Field;

**import** lombok.\*;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

@EqualsAndHashCode

@ToString

@Builder

**public** **class** BouteilleId **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Field(name = "bottle\_id")

**private** **int** idBouteille;

@Field(name = "region\_id")

**private** **int** idRegion;

@Field(name = "color\_id")

**private** **int** idCouleur;

}

* + Pour respecter les noms en base, utilisation de @Field
* La classe Bouteille à 2 attributs :
  + id qui est de type BouteilleId
  + nom
* Il faut poser les annotations de Lombok pour respecter le design pattern POJO
  + @Data
  + Et ajouter @Builder, @NoArgsConstructor et @AllArgsConstructor pour créer des instances facilement
* Pour que Bouteille devienne un Document de MongoDB, il faut
  + @Document + préciser le paramètre collection en bottles
  + @Id sur l’attribut id pour définir l’identifiant
  + @Field sur nom pour le renommer name

**package** fr.eni.gestionavis.bo.vin;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.mongodb.core.mapping.\*;

**import** lombok.\*;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

@Document(collection = "bottles")

**public** **class** Bouteille {

@Id

**private** BouteilleId id;

@Field(name = "name")

**private** String nom;

}

# Créer AvisRepository

**package** fr.eni.gestionavis.dal;

**import** org.springframework.data.mongodb.repository.MongoRepository;

**import** fr.eni.gestionavis.bo.vin.\*;

**public** **interface** BouteilleRepository **extends** MongoRepository<Bouyteille, BouteilleId>{

}

* Faire attention à bien définir le type de l’identifiant comme un type BouteilleId

# Créer des tests unitaires

* En vous inspirant des tests des démonstrations.
* Créer une classe de tests unitaires qui s’appuie sur le contexte de Spring
  + @Slf4j
  + @SpringBootTest
  + @TestMethodOrder(MethodOrderer.MethodName.**class**)
* Créer le test :
  + Sauver une Bouteille et tracer la

**package** fr.eni.gestionavis.clefcomposite;

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** java.util.Optional;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.context.SpringBootTest;

**import** fr.eni.gestionavis.bo.vin.\*;

**import** fr.eni.gestionavis.dal.BouteilleRepository;

**import** lombok.extern.slf4j.Slf4j;

@Slf4j

@SpringBootTest

**class** TestBouteilleDocument {

@Autowired

BouteilleRepository bouteilleRepository;

@Test

**void** test\_save\_bouteille() {

BouteilleId bouteilleId = BouteilleId

.*builder*()

.idBouteille(2298)

.idRegion(5)

.idCouleur(1)

.build();

Bouteille bouteille = Bouteille

.*builder*()

.id(bouteilleId)

.nom("Vin Blanc ENI")

.build();

bouteilleRepository.save(bouteille);

//Vérification en base

Optional<Bouteille> opt = bouteilleRepository.findById(bouteilleId);

*assertThat*(opt).isNotNull();

*assertThat*(opt.isPresent()).isTrue();

Bouteille bouteilleDB = opt.get();

*assertThat*(bouteilleDB).isNotNull();

*assertThat*(bouteilleDB.getNom()).isNotNull();

***log***.info(bouteilleDB.toString());

}

}

* Le test est à vert
* Voici le type de trace obtenu :

Saving Document containing fields: [\_id, name, \_class]

findOne using query: { "id" : { "$java" : BouteilleId(idBouteille=2298, idRegion=5, idCouleur=1) } } fields: Document{{}} for class: class fr.eni.gestionavis.bo.vin.Bouteille in collection: bottles

Bouteille(id=BouteilleId(idBouteille=2298, idRegion=5, idCouleur=1), nom=Vin Blanc ENI)