Gestion des avis (partie 03)

# TP03 du module 03 – Spring Data MongoDB

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Création d’une association embarquée (stratégie 1) * 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 un schéma de l’objet JSON représentant un Document de la Collection reviews en base :

{

\_id: ObjectId('64be4b535e89597fc7bfbd84'),

note: 4,

commentary: 'Doux. A déguster frais',

date: ISODate('2023-07-24T09:58:43.208Z'),

client: {

login: 'bobeponge@email.fr',

quantity\_ordered: 11

},

\_class: 'fr.eni.gestionavis.bo.Avis'

}

Solution

# Créer la classe Client et l’association avec Avis

* Le package des classes est fr.eni.gestionavis.bo
* Utiliser les annotations de Spring Data MongoDB
* La classe Client doit respecter le design pattern POJO
  + Elle s’appelle BouteilleId
  + Elle a 2 attributs qui composent la clef
    - pseudo
    - quantiteCommandee
  + Ajouter les annotations Lombok : @Data, @AllArgsConstructor, @NoArgsConstructor et @Builder

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

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

**import** lombok.\*;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

**public** **class** Client {

@Field(name = "login")

**private** String pseudo;

@Field(name = "quantity\_ordered")

**private** **int** quantiteCommandee;

}

* + Pour respecter les noms en base, utilisation de @Field
* Association embarquée entre Avis et Client
  + Client fera donc parti du Document Avis en base
  + Il sera un sous objet JSON embarqué
* Dans Avis
  + Ajouter un attribut de type Client
  + Et c’est tout pour gérer cette association

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

**import** java.time.LocalDateTime;

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

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

**import** lombok.\*;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

@Document(collection = "reviews")

**public** **class** Avis {

@Id

**private** String id;

**private** **int** note;

@Field(name = "commentary")

**private** String commentaire;

**private** LocalDateTime date;

**private** Client client;

}

# 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 un test :
  + Sauver un Avis avec son Client et tracer le

**package** fr.eni.gestionavis.association;

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

**import** java.time.LocalDateTime;

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

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

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

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

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

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

@Slf4j

@SpringBootTest

@TestMethodOrder(MethodOrderer.MethodName.**class**)

**class** TestAssociationAvisClient {

@Autowired

AvisRepository avisRepository;

@Test

**void** test01\_save\_avis\_stagiaire() {

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.quantiteCommandee(11)

.build();

**final** Avis avis = Avis

.*builder*()

.note(4)

.commentaire("Doux. A déguster frais")

.date(LocalDateTime.*now*())

.build();

//Association

avis.setClient(client);

**final** Avis avisDB = avisRepository.save(avis);

// Vérifier que l'identifiant n'est pas nul

*assertThat*(avisDB.getId()).isNotNull();

*assertThat*(avisDB.getId()).isNotBlank();

// Vérifier que la référence embarquée existe

**final** Client stagiaireDB = avisDB.getClient();

*assertThat*(stagiaireDB).isNotNull();

*assertThat*(stagiaireDB).isEqualTo(client);

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

}

}

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

Inserting Document containing fields: [note, commentary, date, client, \_class] in collection: reviews

Avis(id=64be4b535e89597fc7bfbd84, note=4, commentaire=Doux. A déguster frais, date=2023-07-24T11:58:43.208328500,

client=Client(pseudo=bobeponge@email.fr, quantiteCommandee=11))