Gestion des avis (Partie 07)

# TP02 du module 05 – Spring Data REST

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Manipulation de requêtes spécifiques |

|  |
| --- |
| **Durée estimée** |
| 30 à 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 vins.

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

Dans les itérations de ce module, nous allons développer l’API Web.

Solution

# Réalisation :

Les niveaux 1 sont obligatoires,

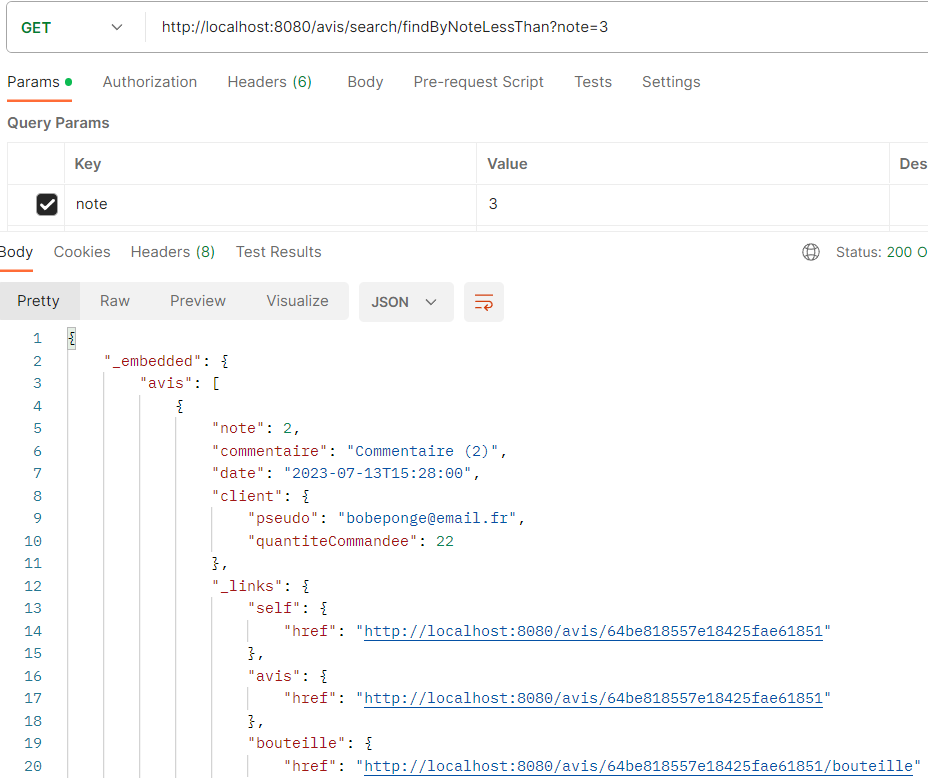
Les niveaux 2 sont plus poussés et optionnels

# Niveau 1 : Appel des requêtes spécifiques avec paramètres simples

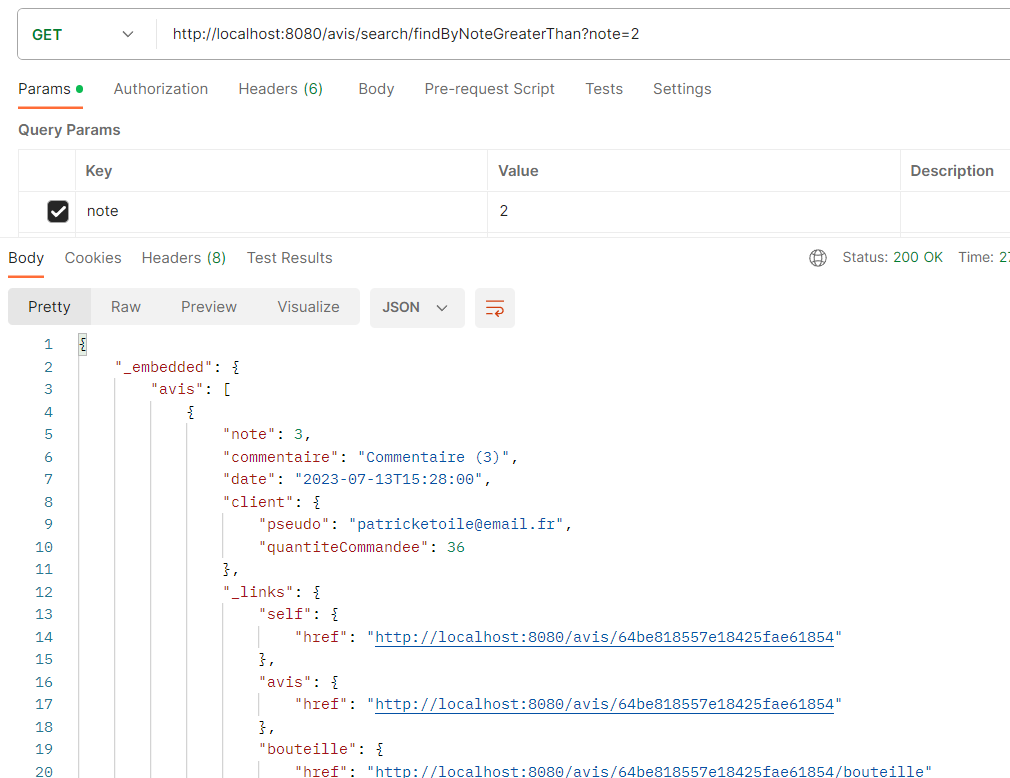
* Tester les 2 méthodes de recherche sur la note des Avis
  + Aucun code n’est à ajouter car nous avons déjà déclaré les méthodes dans le module 03
* Il suffit de créer des URL avec dans le path

/search/"nom\_méthode"?"nom\_paramètre"

* Les URLs sont très similaires
  + La racine est la même : http://localhost:8080/avis/search/
  + Le paramètre a le même nom : ?note=…
  + Le nom de la méthode change :
    - findByNoteLessThan
    - findByNoteGreaterThan
* Les résultats dépendent de la méthode
  + findByNoteLessThan :

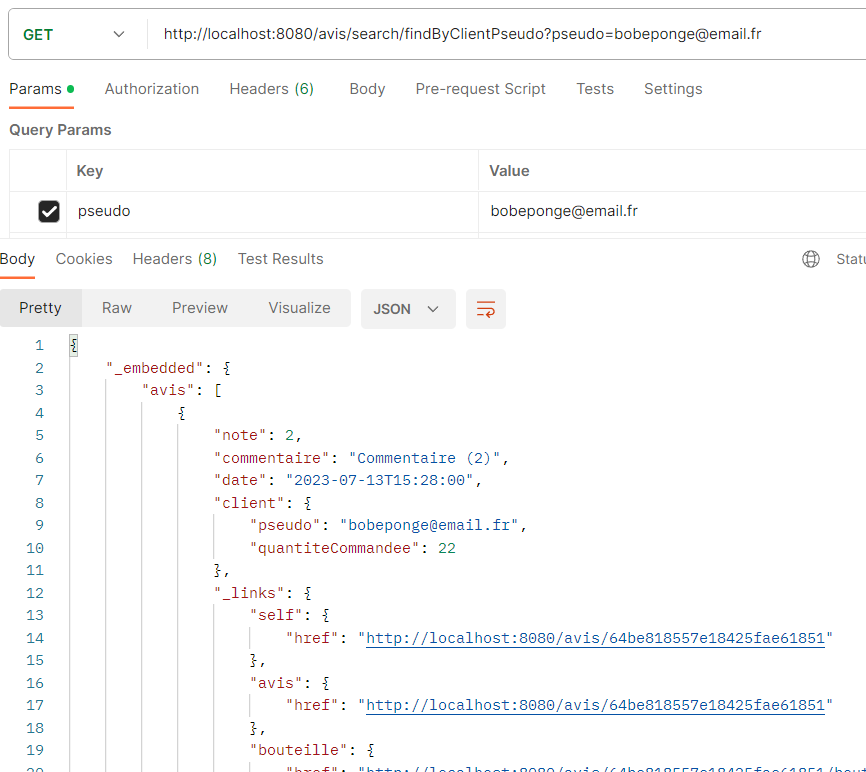


* + findByNoteGreaterThan :

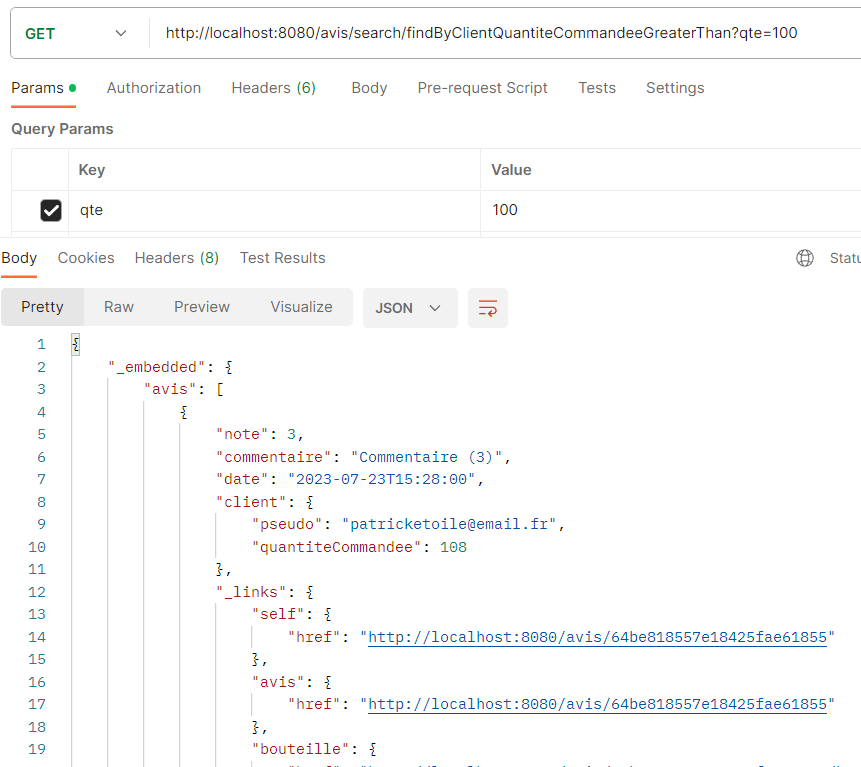


# Niveau 1 : Appel des requêtes spécifiques sur des associations

* Dans AvisRepository, tester la méthode de findByClientPseudo
  + « pseudo » est un champ unique et simple (il correspond à une clef primaire pour MongoDB) pour Client
  + Donc l’association Avis-Client peut manipuler directement cette méthode de requête spécifique



* Dans AvisRepository, tester la méthode de findByClientQuantiteCommandeeGreaterThan
  + « quantite » est un champ de type simple.
  + Donc l’association Avis-Client peut manipuler directement ce champ via la méthode de requête spécifique



# Niveau 2 : Clef composite et requête spécifique

* Une clef composite est un objet complexe
  + Il faudrait passer un JSON dans l’URL
  + Ce n’est pas possible
  + Il n’est donc pas envisageable de travailler sur des clefs de ce type
* Il faut donc repenser son projet :
  1. Soit ; mettre en place un @RestController et un @Service pour manipuler la méthode du Repository en passant les données de la clef composite en plusieurs paramètres dans l’URL
  2. Soit ; repenser les BO pour manipuler des clefs primaires simples
* Pour manipuler la méthode findByBouteille : il faut travailler sur l’association Avis-Bouteille
  + Le Document Bouteille a une clef composite
  + Demandez-vous qu’elle solution est viable pour manipuler cette méthode entre les 2 choix
  + Et mettre en place votre solution.
* **Nous allons faire le choix 2.**
  + Nous sommes en mode création d’un nouveau projet, il n’y a pas de base de données en production
  + Il est donc possible de modifier le schéma de la base et des BO

## Suppression de la clef composite

* D’après le métier du projet cave à vin ; nous pouvons valider que « idBouteille » est un élément unique
  + Il suffit comme identifiant pour les Bouteille
* Modifier la classe Bouteille comme suit :

**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

@Field(name = "bottle\_id")

**private** Integer idBouteille;

@Field(name = "region\_id")

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

@Field(name = "color\_id")

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

@Field(name = "name")

**private** String nom;

}

* + Son identifiant est « idBouteille », il est de type Integer pour permettre les transtypages entre String et Integer plus facilement
  + « idRegion » et « idCouleur » deviennent de simples attributs
  + Supprimer la classe BouteilleId
* Modifier BouteilleRepository, la clef primaire est de type Integer maintenant

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

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

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

**public** **interface** BouteilleRepository **extends** MongoRepository<Bouteille, Integer>{

}

## Recherche par Bouteille

* Nous devons remplacer dans AvisRepository la méthode « findByBouteille » qui manipule un objet complexe par findByBouteilleBouteilleId

…

@RepositoryRestResource(collectionResourceRel = "avis", path = "avis")

**public** **interface** AvisRepository **extends** MongoRepository<Avis, String> {

…

// Remonter les Avis associé à une Bouteille

//List<Avis> findByBouteille(@Param("bouteille") Bouteille bouteille);

List<Avis> findByBouteilleIdBouteille(@Param("idBouteille") Integer idBouteille);

…

}

## Mise à jour des tests unitaires

* Pour respecter l’évolution du code ; il faut revoir les tests unitaires et la base de données de notre application
* Supprimer la base de données et la recrée. Pour repartir d’une base vierge (Gestion\_Avis)
* Déplacer la classe TestBouteilleDocument dans le package fr.eni.gestionavis.bo
  + Supprimer le package fr.eni.gestionavis.bo.clefcomposite
  + Modifier le test pour que la clef soit idBouteille

@Test

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

Integer bouteilleId = 2298;

Bouteille bouteille = Bouteille

.*builder*()

.idBouteille(bouteilleId)

.idRegion(5)

.idCouleur(1)

.nom("Vin Blanc ENI")

.build();

bouteilleRepository.save(bouteille);

…

}

* Modifier la classe de tests d’association avec Avis-Bouteille : TestAssociationAvisBouteille

// Vérifier que le Bouteille est complet

*assertThat*(avisDB.getBouteille().getIdBouteille()).isGreaterThan(0);

*assertThat*(avisDB.getBouteille()).isEqualTo(bouteilleDB);

* Modifier la classe de création de données : DataPourRequetes

**void** insertion\_Bouteille\_DB() {

**final** List<Bouteille> listeBouteilles = **new** ArrayList<>();

// Création de 3 Bouteille

listeBouteilles.add(Bouteille

.*builder*()

.idBouteille(18298)

.idRegion(3)

.idCouleur(1)

.nom("Vin ENI Edition")

.build());

listeBouteilles.add(Bouteille

.*builder*()

.idBouteille(1298)

.idRegion(3)

.idCouleur(2)

.nom("Vin ENI Service")

.build());

listeBouteilles.add(Bouteille

.*builder*()

.idBouteille(1999)

.idRegion(2)

.idCouleur(3)

.nom("Vin ENI Ecole")

.build());

listeBouteilles.forEach(b -> {

bouteilleRepository.save(b);

});

}

* Modifier la classe de tests des requêtes : TestRequetes

@Test

**void** test03\_findByBouteille() {

Integer idBouteille = 18298;

List<Avis> listeAvis = avisRepository.findByBouteilleIdBouteille(idBouteille);

*assertThat*(listeAvis).isNotNull();

*assertThat*(listeAvis).isNotEmpty();

*assertThat*(listeAvis.size()).isEqualTo(3);

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

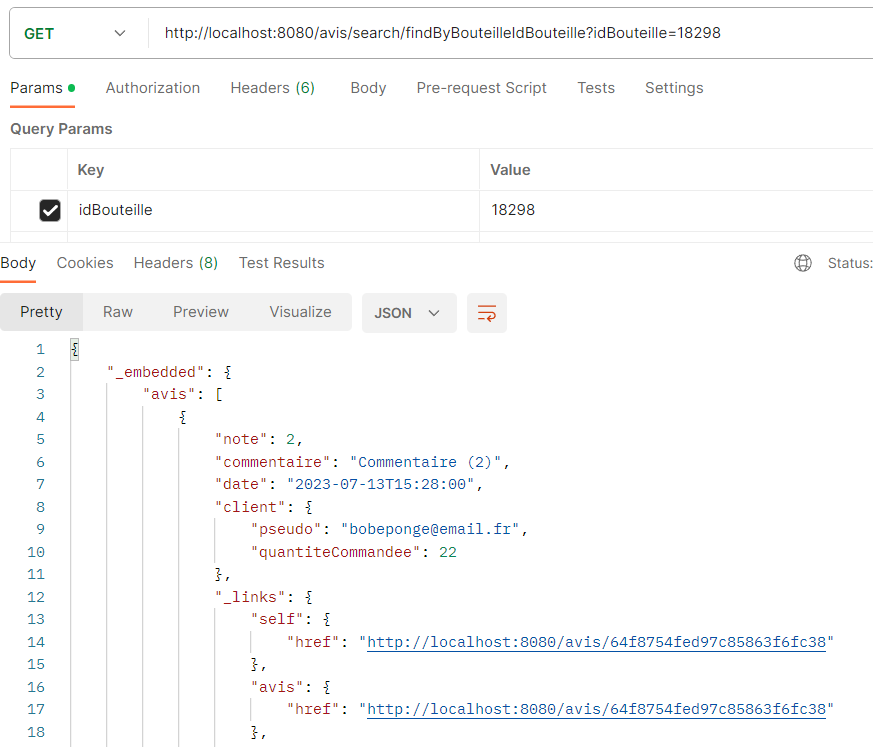
}

## Exécution de l’ensemble des tests

* Il est possible de lancer en une fois tous les tests unitaires
  + Faire un clic droit sur « src/test/java »
  + Run As 🡪 JUnit Test
* Tous les tests seront verts

## Exécution

* Relancer votre application
* Tester la requête de recherche par cours
* http://localhost:8080/avis/search/findByBouteilleIdBouteille?idBouteille=18298



# Niveau 2 : LocalDateTime et requête spécifique

* Pour utiliser la méthode findByDateBetween ; il faut pouvoir passer des LocalDateTime par l’URL
* Il s’agit d’objet, mais il y a une solution en utilisant l’annotation @DateTimeFormat
  + Voici le lien vers la documentation de Spring :

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/format/annotation/DateTimeFormat.html>

* + Cette annotation permet de préciser le format de la date transmise. Dans un URL, une date reste une chaîne de caractères, il faut que Spring la convertisse
  + Dans notre cas, nous allons préciser le format : DateTimeFormat.ISO.DATE\_TIME
  + Et ainsi, dans l’URL ; il faudra passer la date en yyyy-MM-dd HH:mm:ss

**…**

**import** org.springframework.format.annotation.DateTimeFormat;

**…**

@RepositoryRestResource(collectionResourceRel = "avis", path = "avis")

**public** **interface** AvisRepository **extends** MongoRepository<Avis, String> {

…

// Remonter les Avis dont la date est entre les 2 dates en paramètres

// Pour gérer une date au travers de Spring Data REST utiliser @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME)

List<Avis> findByDateBetween(

@Param("deb") @DateTimeFormat(iso = DateTimeFormat.ISO.***DATE\_TIME***) LocalDateTime deb,

@Param("fin")@DateTimeFormat(iso = DateTimeFormat.ISO.***DATE\_TIME***) LocalDateTime fin);

}

* Mettre en place la solution et la tester avec l’URL :

http://localhost:8080/avis/search/findByDateBetween?deb=2023-07-13T00:00:00.000Z&fin=2023-07-31T00:00:00.000Z

