Gestion des avis (Partie 06)

# TP01 du module 05 – Spring Data REST

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Ajout de la configuration Spring Data REST * Utilisation de @RepositoryRestResource pour les URL |

|  |
| --- |
| **Durée estimée** |
| 30 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.

* Si vous n’aviez pas fini le TP du module 03. Vous trouverez les classes solutions dans un zip des ressources.

Solution

* Nous allons utiliser Spring Data REST pour avoir une API complète sur notre projet
  + Nous pouvons réaliser cette API car il n’y a pas de métier pour le moment d’associé

Repository

Spring Data REST



Spring Data

DB

# Configuration de Spring Data REST dans build.gradle

* Ajout du starter Spring Data REST

dependencies {

//Starter API REST

implementation 'org.springframework.boot:spring-boot-starter-data-rest'

* Faire « Refresh Gradle Project »

## BO :

Nous avons déjà une couche BO qui met en place les annotations de Spring Data. Il n’y a rien à faire à ce niveau pour le moment.

## DAL :

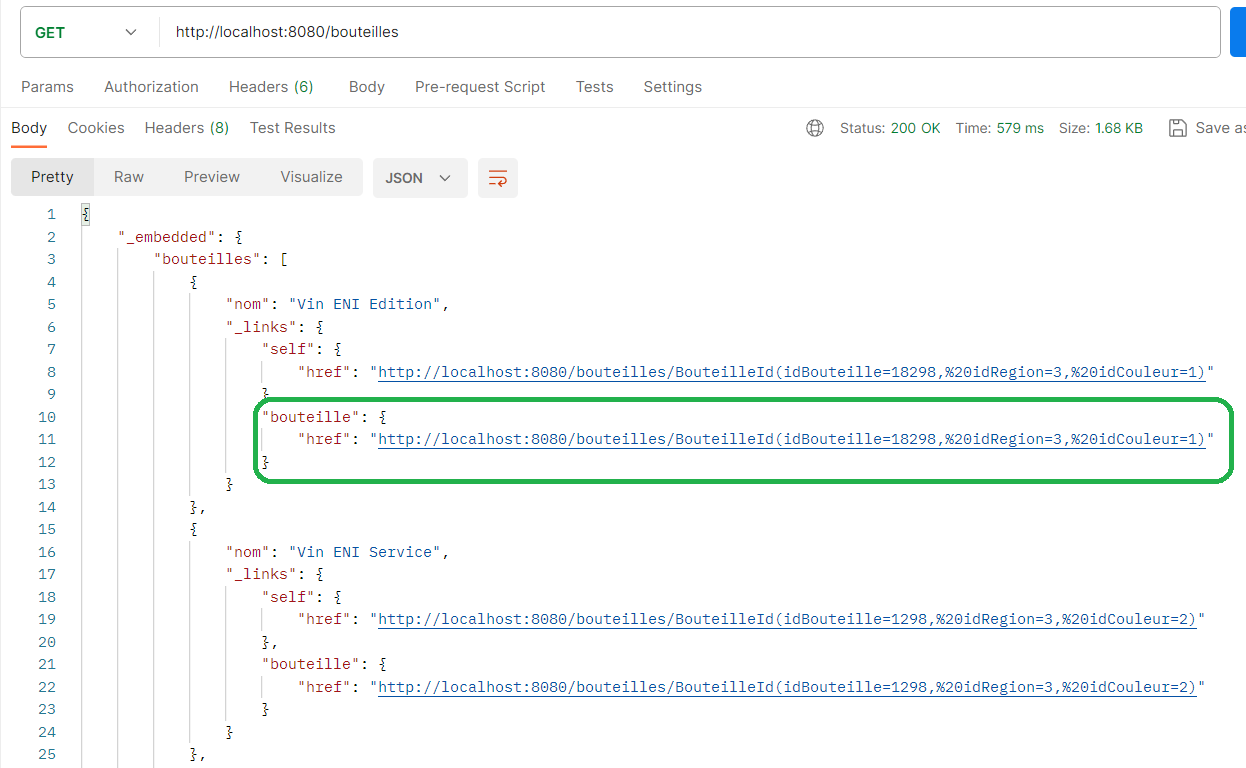
Nous avons déjà une couche DAL qui déclarer pour chaque Document un Repository de Spring Data. Il n’y a rien à faire à ce niveau non plus.

## Contrôleur de l’API REST :

* C’est le module Spring Data REST qui le fournit.
* Il permettra de manipuler des requêtes sur toutes les méthodes du protocole http
* Et même d’utiliser toutes les méthodes de MongoRepository (pagination, tri, …)
* Par défaut Spring utilise le nom de la classe au pluriel en anglais pour créer l’alias des urls.

## Exécution avec Postman

* Lancer l’application de votre serveur.
* Lancer Postman
* Tester l’API pour Bouteille
  + L’URL par défaut sera : http://localhost:8080/bouteilles
* Dans Postman, cliquer pour créer une nouvelle requête GET dessus



* Spring Data REST a généré le contrôleur pour répondre à la demande
* L’utilisation de l’API est assez simple et intuitive.
* De plus, les informations HATEOAS fournies par Spring Data REST vous permettent de cibler et d’automatiser l’utilisation de l’API par les clients.
  + Dans les informations présentées, nous avons directement l’URL pour manipuler Bouteille par son identifiant
  + Si vous descendez tout en bas du body de la réponse
    - Remarquez qu’il nous informe :
    - Du nombre d’éléments total : 3
    - Du nombre de pages : 1
    - Le nombre d’éléments par page : 20
    - Il propose 2 URLs :
      * Le premier permet de gérer la pagination et le nombre d’éléments par page
      * Le deuxième, permet de voir le profil du Bouteille au format ALPS

# Définition des ressources via @RepositoryResource

* Définir un nom de ressources correctes pour Avis
  + Nous voulons que les noms de la ressource et dans le path soient : avis
  + Et pas avises comme proposé par défaut par Spring Data REST
* Modifier AvisRepository pour que les noms de la ressource et dans le path soient : avis

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

**…**

**import** org.springframework.data.rest.core.annotation.RepositoryRestResource;

…

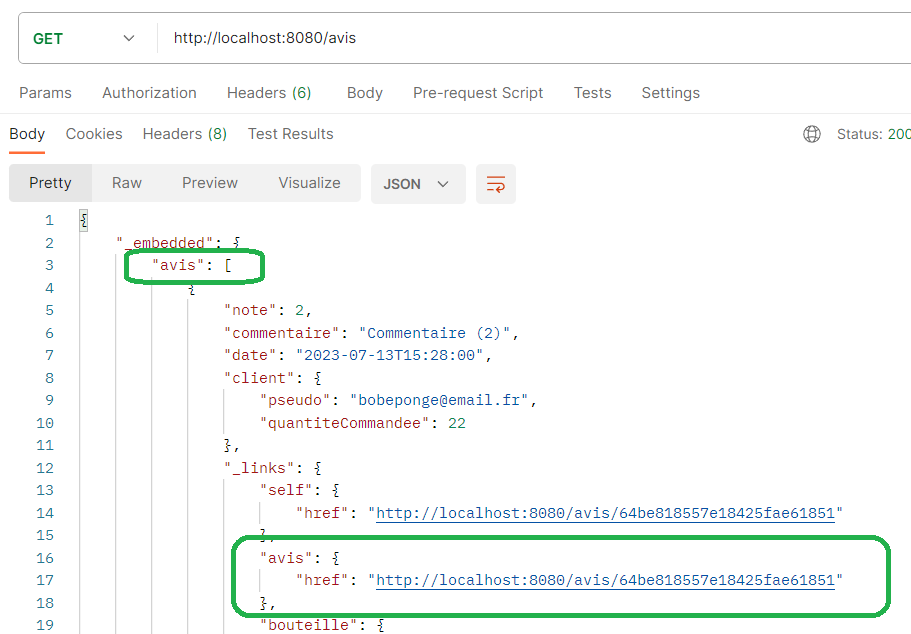
@RepositoryRestResource(collectionResourceRel = "avis", path = "avis")

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

…

}

## Tester avec Postman



* Toutes les URLs sont maintenant avec une racine « avis » pour ce Repository
* Et le nom de la ressource est bien « avis » aussi.