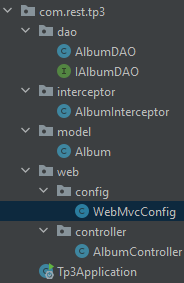
Spring Boot - Micro-services

Reynault Sies

Voici à quoi ressemble l’arborescence du projet, dans ce rapport, je vais expliquer les éléments qui la composent point par point :



# Création du DAO

Pour créer le DAO, j’ai commencé la mise en place d’une interface indiquant toutes les opérations que je souhaite implémenter et partager au contrôleur de la manière suivante :

On a également son implémentation concrète avec une collections contenant l’ensemble des modèles qui sont ici, des albums. On peut également voir l’utilisation de l’annotation @Repository qui permet d’indiquer la classe qui gère les données persistées.

# 

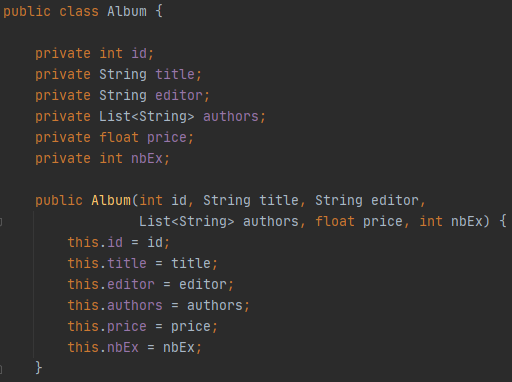
# 

Voici, par exemple, la méthode qui réalise l’insertion d’un album. J’ai imité le comportement d’une base de données en ajoutant une variable qui s’incrémente au fur et à mesure de l’ajout des Albums, chose qui évidemment sera remplacé par un identificateur qui s’incrémente de manière automatique dans une base de données.

Si on regarde les autres méthodes, on a globalement les éléments classiques de récupération et de suppression de données. On peut par exemple voir que la méthode put vérifie que l’album ajouté existe déjà.

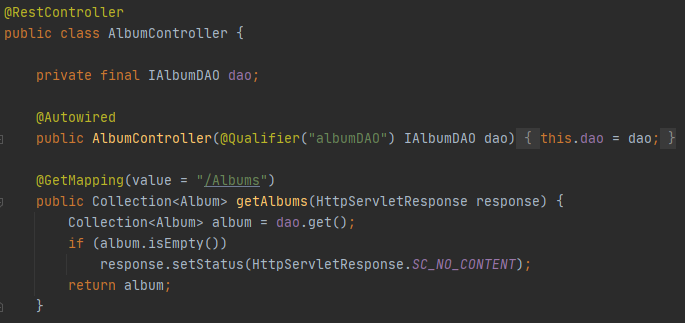
# Création du Modèle

Pour le modèle, j’ai choisi de représenter un Album avec un titre, un id, un prix, un nombre d’exemplaires, un éditeur et une liste d’auteurs. La classe est simplement un POJO avec getters et setters.



# Création du Contrôleur

Enfin, on créer la classe du contrôleur qui va récupérer les différentes requêtes des clients.



On peut voir l’utilisation de l’annotation @RestController qui permet d’indiquer à Spring que notre classe est un contrôleur qui va récupérer les requêtes du client. Ici, pas de chemin de base associé à la classe, mais un chemin pour chaque méthode. Par exemple, on peut voir que la méthode getAlbums permet de récupérer la liste des albums et vérifie si celle-ci est vide ou non, si c’est le cas elle indique au client qu’il n’y a pas de contenu en modifiant le Header de la réponse.

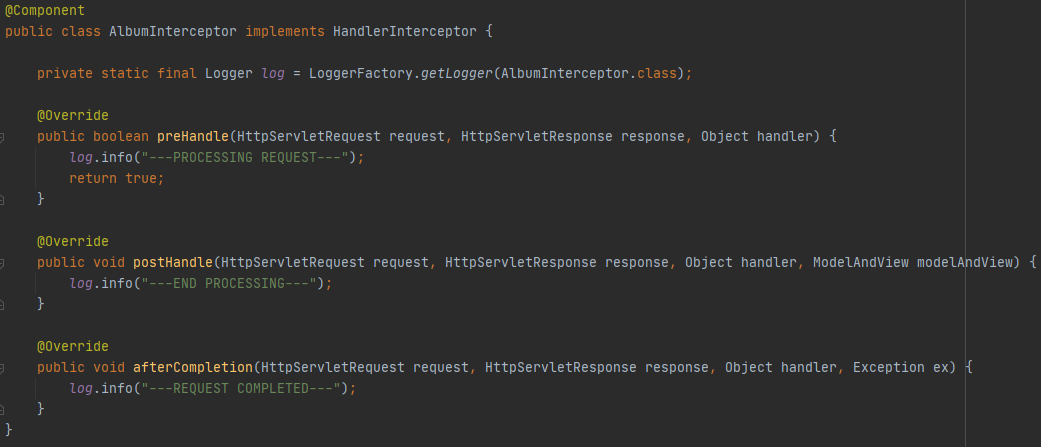


On peut également voir que les autres méthodes utilisent de la même manière le service DAO qui contient les données. Donc si on voulait remplacer par une base de données, cela se ferait juste en remplaçant le qualifier associé à l’injection du DAO dans le constructeur. Nous n’aurions pas à toucher la classe.

# Mise en place d’intercepteurs

Pour vérifier les entrées utilisateurs et les sorties du service /Albums on peut utiliser un intercepteur qui exécutera une méthode pour chaque entrée et sortie des méthodes du contrôleur. J’ai mis en place ce système pour par exemple utiliser un système de log qui indique au serveur quand il y a une requête du client vers ce service. Pour ce faire on procède de la manière suivante :

* Création de la classe qui servira d’intercepteur en implémentant une interface.
* Il faut alors implémenter chaque fonction, avant le contrôleur, après et après la génération de la vue.



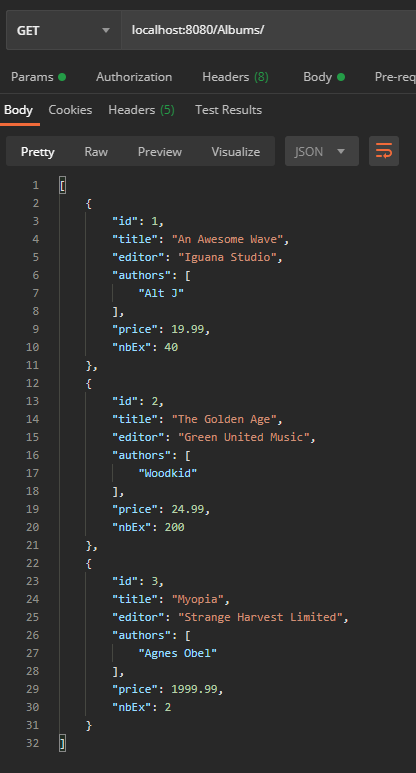
* Il faut ensuite indiquer à Spring Boot une configuration additionnelle en ajoutant cet intercepteur à la bonne route, ici j’ai ajouté l’intercepteur pour les routes « /Albums » et « /Albums/{id} ».



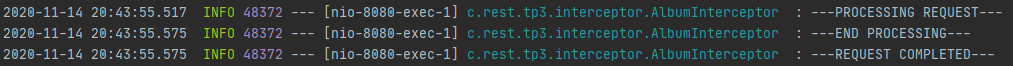
# Vérifications avec Postman

Il faut ensuite vérifier les différentes routes avec Postman :

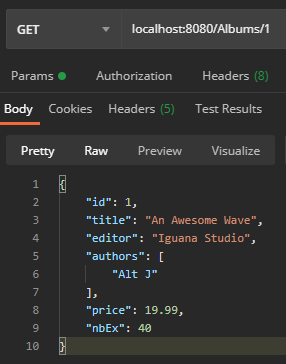
* GET (/Albums) :



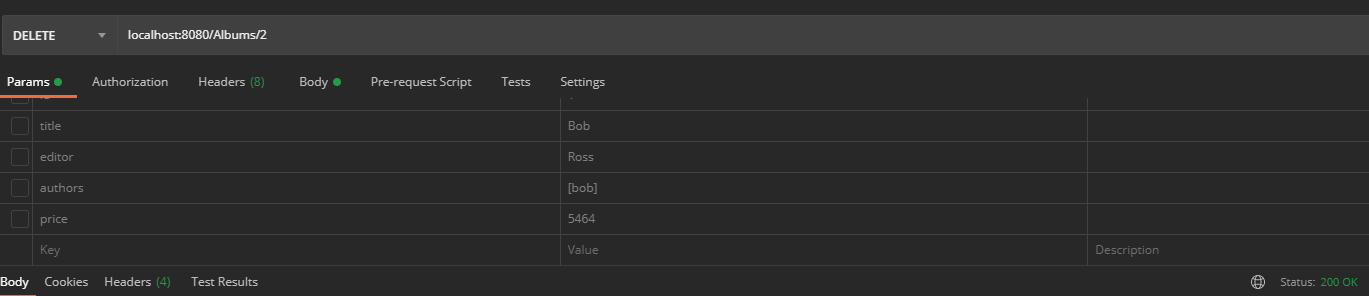
On peut également voir les logs dans le terminal du serveur.



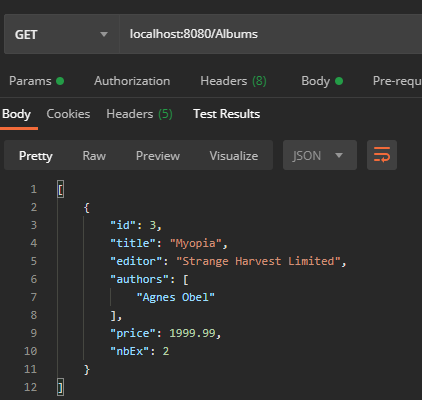
* GET (/Albums/1) :



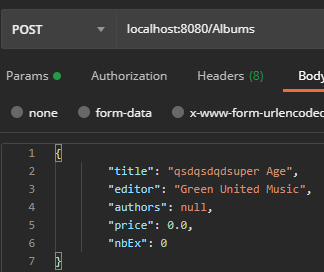
* DELETE (/Albums/1) et DELETE (/Albums/2) :



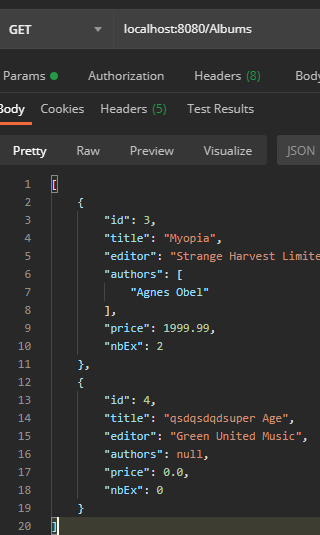
Avec l’affichage de la liste qui ne contient plus qu’un album :



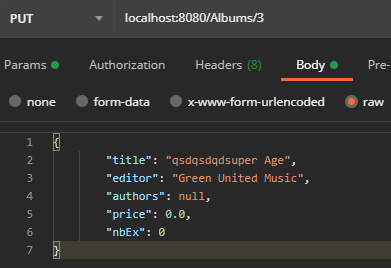
* POST( /Albums) :



Puis affichage de la liste qui doit désormais contenir un autre album :



* PUT (/Albums/3) :



Affichage de la liste des éléments :

