

Projet JEE/Spring I

---

# Sujet 2, Gestion de discussions

---

Filière Informatique et systèmes de communication

Orientation Développement logiciel.

Réalisé par

Noa Devanthéry

Présenté à

Chèvre Sébastien

## Table des matières

<b>Introduction</b> .....	3
<b>Contexte</b> .....	3
Architecture implémentée .....	3
Model .....	4
Repository .....	4
Service .....	4
Controller.....	5
<b>Problèmes, résolutions et choix</b> .....	6
Récursion infinie .....	6
Problème de suppression .....	6
Planning prévu et effectif .....	7
Bilan .....	8
Bibliographie.....	8

## Introduction

### Contexte

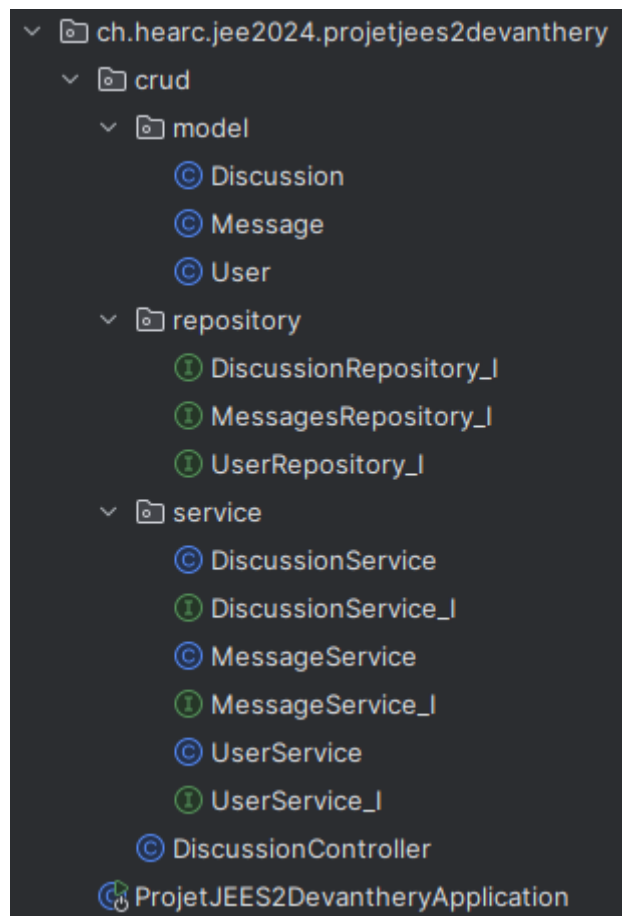
Dans le cadre du cours JEE/Spring I, nous avons étudié les principes du développement d'applications web modernes en utilisant Spring Boot. Ce projet vise à mettre en pratique les concepts appris, notamment la gestion des API REST, la persistance des données avec JPA, et la gestion de tests.

Le projet consiste à développer une application de gestion de discussions et de messages, offrant des fonctionnalités telles que la création, la mise à jour, la suppression et la récupération de discussions et de messages.

L'objectif principal est d'appliquer les bonnes pratiques du développement logiciel, comme la séparation des couches (controller, service, repository), la gestion des statuts HTTP appropriés ainsi que l'implémentation de tests automatisés.

### Architecture implémentée

L'application est organisée selon une architecture en couches, qui sépare les différentes parties du projet (modèle, service, repository et contrôleur).



## Model

Le modèle représente les entités de l'application qui sont mappées à la base de données grâce à JPA. Les entités principales de ce projet sont :

- **Discussion** : Représente une discussion avec comme champ : id, name, subject et une liste de messages.
- **Message** : Représente un message lié à une discussion, avec comme champs : id, content, date, et une relation bidirectionnelle avec une Discussion.

Ces entités utilisent des annotations comme @Entity, @Table, @OneToMany, et @ManyToOne pour définir les relations et les contraintes de persistance.

## Repository

La couche repository est responsable de la communication avec la base de données. Elle est implémentée à l'aide des interfaces Spring Data JPA :

- **DiscussionRepository\_I** : Fournit des méthodes pour accéder aux données des discussions.
- **MessagesRepository\_I** : Permet d'effectuer des opérations CRUD sur les messages.

Ces interfaces héritent de CrudRepository. Ce qui permet de gérer automatiquement les différentes requêtes des services.

## Service

La couche service contient la logique métier et agit comme un intermédiaire entre le contrôleur et les repositories.

- **DiscussionService** :
  - Gère les discussions : création, mise à jour, suppression et récupération.
  - Vérifie l'existence d'une discussion avant de permettre des modifications ou suppressions.
- **MessageService** :
  - Assure la gestion des messages associés aux discussions.
  - Ajoute un message à une discussion existante ou met à jour son contenu.

Chaque service utilise des exceptions pour signaler des erreurs (ex. : discussion ou message introuvable).

## Controller

Le **DiscussionController** regroupe les endpoints REST pour gérer à la fois les **discussions** et les **messages associés**. Les fonctionnalités implémentées incluent :

### Discussions :

- Récupération d'une discussion par ID (GET /discussions/{id}).
- Création d'une nouvelle discussion (POST /discussions).
- Mise à jour d'une discussion existante (PUT /discussions/{id}).
- Suppression d'une discussion (DELETE /discussions/{id}).

### Messages :

- Récupération des messages d'une discussion (GET /discussions/{id}/messages).
- Ajout d'un message à une discussion (POST /discussions/{id}/messages).
- Modification d'un message (PUT /discussions/{discussionId}/messages/{messageId}).
- Suppression d'un message (DELETE /discussions/{discussionId}/messages/{messageId}).

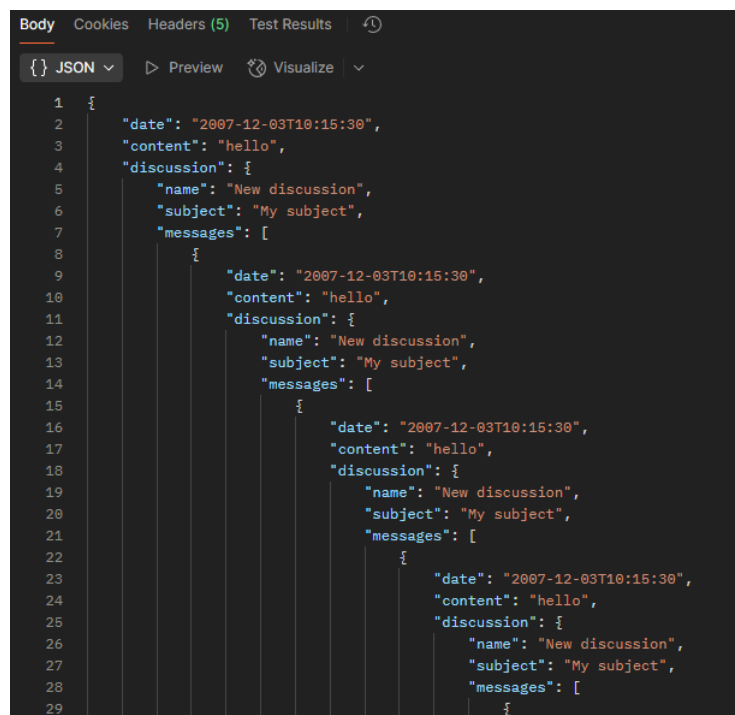
Le contrôleur retourne des réponses HTTP avec des statuts appropriés (200 OK, 201 Created, 404 Not Found, etc.) pour informer du résultat de chaque opération.

Il a été choisi de faire un seul controller pour les discussions et les messages. Cela permet de faciliter la gestion des messages et d'alléger la structure du projet, car les messages n'existent pas sans discussions.

## Problèmes, résolutions et choix

### Récursion infinie

Lors du GET sur une discussion après avoir implémenté et ajouté les messages aux discussions, un problème de récursion infini est apparu. Etant donné que ces entités sont liées de manière bidirectionnelle : un **Message** est associé à une **Discussion**, et une **Discussion** peut contenir plusieurs **Messages**. Lors d'un GET pour récupérer une discussion avec ses messages, ou inversement, chaque entité tentait de sérialiser l'autre, ce qui entraînait une boucle infinie.



```
1 {
2   "date": "2007-12-03T10:15:30",
3   "content": "hello",
4   "discussion": {
5     "name": "New discussion",
6     "subject": "My subject",
7     "messages": [
8       {
9         "date": "2007-12-03T10:15:30",
10        "content": "hello",
11        "discussion": {
12          "name": "New discussion",
13          "subject": "My subject",
14          "messages": [
15            {
16              "date": "2007-12-03T10:15:30",
17              "content": "hello",
18              "discussion": {
19                "name": "New discussion",
20                "subject": "My subject",
21                "messages": [
22                  {
23                    "date": "2007-12-03T10:15:30",
24                    "content": "hello",
25                    "discussion": {
26                      "name": "New discussion",
27                      "subject": "My subject",
28                      "messages": [
29                        {
```

Pour résoudre ce souci, il a fallu utiliser `@JsonManagedReference` et `@JsonBackReference` de la bibliothèque Jackson, qui permet de spécifier quelle partie de la relation bidirectionnelle doit être sérialisée et laquelle doit être ignorée.

Donc dans la classe **Discussion**, il faut annoter la liste des messages avec `@JsonManagedReference` et dans la classe **Message**, il faut annotée avec `@JsonBackReference`.

Cela a permis d'éviter la récursion infinie en désignant explicitement les relations à sérialiser dans la réponse API.

### Problème de suppression

Le problème de suppression s'est posé lors de la tentative de suppression d'une discussion contenant des messages associés. Par défaut, la base de données empêche la suppression d'une entité si elle contient des relations (par exemple, les messages associés à une discussion). Ce qui donnait une erreur.

Pour résoudre ce problème, 2 solutions ont été explorées :

1. **Suppression en cascade** : En modifiant le @OneToMany en @OneToMany(cascade = CascadeType.ALL) sur la relation entre **Discussion** et **Message**. Cela a permis de supprimer les messages associés lorsqu'une discussion est supprimée et ça garantit que les messages seront automatiquement supprimés de la base de données lorsqu'une discussion est supprimée, sans avoir besoin de changer le code.
2. **Suppression manuelle des messages** : Une autre approche consistait à supprimer explicitement les messages associés avant de supprimer la discussion. Cette solution fut ajoutée dans le service **DiscussionService**, où tous les messages associés étaient supprimés avant la discussion.

Les deux approches ont été testées et logiquement la solution la plus simple, la suppression en cascade, a été choisie.

## Planning prévu et effectif

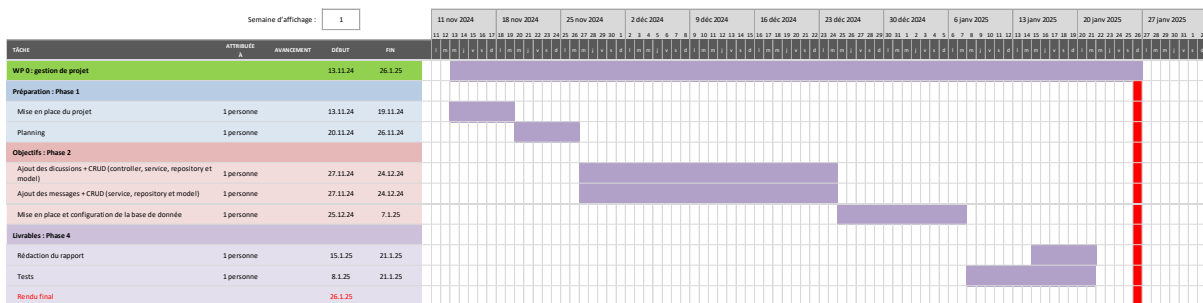
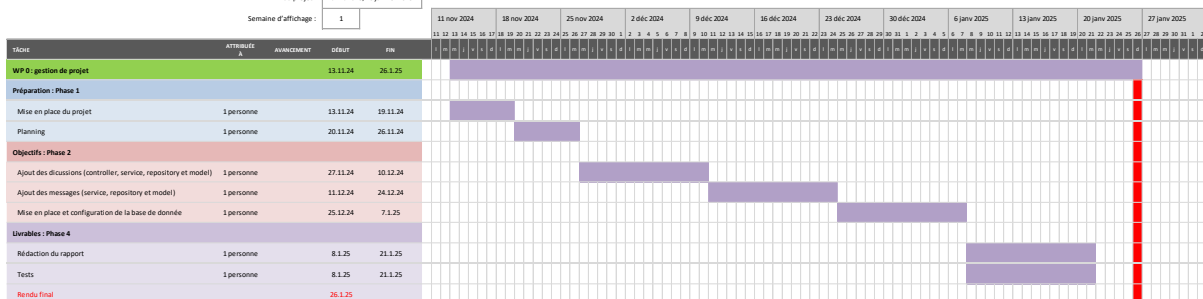
### Sujet 2 - Gestions de discussion

Devanthery Noa  
HE-Arc ISC3-IL

Début du projet : mercredi, 13 novembre 2024

Fin du projet : dimanche, 26 janvier 2025

Semaine d'affichage : 1



## Bilan

Le projet Gestion de discussions, développé dans le cadre du cours JEE/Spring I, a permis d'appliquer : l'architecture en couches, les API REST et la persistance des données avec JPA. L'application fonctionne bien, elle peut gérer les discussions et messages, tout en respectant les bonnes pratiques.

Quelques petits soucis sont apparus, des problèmes de récursion infinie dans les relations entités, résolus grâce à Jackson, et la suppression des entités liées finalement géré par cascade.

## Bibliographie

**Let's Solve the Infinite Recursion Stackoverflow Problem in Spring Boot**, POOJITHA IROSHA, 1 Juin 2023 :

<https://medium.com/@poojithairosha/lets-solve-the-infinite-recursion-stackoverflow-problem-in-spring-boot-8a0a86cf0aba>

**Spring: Trouble understanding cascade deleting entities**, Answered by jwpol, 24 Avril 2022 :

<https://stackoverflow.com/questions/71991459/spring-trouble-understanding-cascade-deleting-entities>