Rapport du projet DiscussionApp

Architecture Implémentée

L'application DiscussionApp a été conçue en suivant une architecture **n-tier** basée sur les principes de **Spring Boot**. Les différentes couches implémentées sont :

1. **Couches de Contrôleurs**
    - Utilisation de RESTful APIs exposées à travers des endpoints pour gérer les entités principales (User, Discussion, Category, Response).
    - Validation des entrées et réponses personnalisées basées sur les statuts HTTP appropriés.
2. **Couche des Services**
    - Gestion de la logique métier, comme la vérification des rôles utilisateur (ADMIN vs USER) pour certaines opérations sensibles.
    - Implémentation des règles comme la création de discussions et la suppression ou le blocage des discussions uniquement par des administrateurs.
3. **Couches de Données**
    - **Repository :** Interaction avec une base de données relationnelle h2 via Spring Data JPA.
    - **Model :** Utilisation d'entités comme User, Category, Discussion et Response avec des relations bien définies (@OneToMany, @ManyToOne).
4. **Pagination et tri**
    - Les endpoints REST permettent la pagination et le tri (par exemple, /api/discussions?page=0&size=10&sort=title).

Problèmes, Résolutions et Choix

Au départ, le projet devait intégrer **Spring Security** pour la gestion des autorisations et des rôles utilisateur (ADMIN vs USER). Cependant, en raison des contraintes de temps et de la priorité donnée à d'autres fonctionnalités comme la pagination et les tests, cette solution n'a pas été implémentée.

Choix : Une gestion personnalisée des autorisations a été mise en place. Les vérifications des rôles utilisateur ont été effectuées via des appels explicites à User.getRole() avant d'exécuter des opérations sensibles (par exemple, blocage ou suppression de discussions).

Impact : Cette approche a permis de maintenir le contrôle des autorisations de manière simple et efficace dans le contexte de ce projet. Toutefois, l'intégration future de **Spring**

Security reste recommandée pour simplifier et renforcer la sécurité du projet, notamment pour gérer les sessions, les jetons JWT, ou les connexions utilisateur.

Planning Initial vs Effectif

Tâche	Durée Prévue	Durée Réelle
Conception de l'architecture	1 jours	1 jours
Implémentation des contrôleurs	3 jours	4 jours
Pagination et tri	1 jour	1 jours
Tests unitaires	2 jours	3 jours
Documentation et rapport	1 jour	2 jours

Tests

Une attention particulière a été portée à l'écriture de tests unitaires et d'intégration pour garantir la robustesse et la fiabilité de l'application.

- **Tests unitaires :** Chaque service a été testé pour vérifier la logique métier et les validations des rôles utilisateur (ADMIN vs USER).
- **Tests d'intégration :** Les endpoints REST ont été testés avec MockMvc pour valider les réponses et les statuts HTTP.
- **Couverture des cas principaux :**
    - Création d'entités (utilisateurs, catégories, discussions, réponses).
    - Validation des autorisations pour des opérations sensibles (blocage/suppression de discussions).
    - Gestion de la pagination et du tri.

Bilan

- **Points Positifs**
    - Architecture claire et extensible.
    - Gestion efficace des autorisations sans Spring Security.
    - Implémentation des tests unitaires couvrant les cas principaux.
- **Points À Améliorer**
    - Répartition initiale de la logique métier entre services et contrôleurs.
- **Réflexion Finale** Ce projet a permis d'approfondir la maîtrise de Spring Boot et de ses outils connexes (Spring Data JPA, MockMvc, etc.). La gestion des autorisations personnalisées a été une décision judicieuse pour un projet de cette envergure. Toutefois, l'intégration future de Spring Security pourrait simplifier certaines parties du code. L'expérience acquise est très enrichissante et ouvre la voie à des projets plus complexes.