Rapport - Discussions API REST

Keschubay Jun

The objective of this project was to develop a REST API for a discussion platform with features like managing discussions, categories, and user roles while ensuring security and scalability.

Implemented architecture

Application built using **Spring Boot** with dependency injection for decoupled architecture.

Layers:

- **Controller Layer**: Manages REST API endpoints.
  In specifics, there are Discussion, AppUser and Category controllers
- **Service Layer**: Contains business logic and handles interactions between controllers and repositories.
  In specifics, Discussion, AppUser and Category services
- **Repository Layer**: Handles database interactions using Spring Data JPA.
  Discussion, AppUser, Category, Comment repositories

Database: Relational database for storing users, discussions, categories, and comments.

Categories are the general themes or directions, created by administrators and facilitating searching thematic discussions. Discussions are user-created 'sub-categories' where they can suggest a topic to discuss. Discussions each belong to some category and are created by someone. Discussions have multiple comments left by different users.

(API documentation is available in the README of the repository, endpoints described there)

Problems, resolutions and choices

It was decided not to have Services and controllers for comments, because their existence is closely related to the context of the discussion where they lay, and they don't require separate management (one wouldn't open a comment in a separate window later, as in Reddit).

- Figuring out application structure.
    - The structured approach helped, figuring out what's needed of the database, and what endpoints will be needed to be met. Looking at how other forums work from a user perspective helped.
- The username wasn't originally planned to be unique
    - It was initially planned to have username as non-unique, and Id as primary identifier. It was also planned to have explicit ManyToOne relationship with users for Discussions and Comments. However, to facilitate implementation and authorization, it was decided to change it to simply storing the username of

the creator and making username unique, so that retrieving it from Security Context will be enough to create underlying objects and verifying roles.

- Making tests
    - While it wasn't a problem with Unit tests, it was a consuming task because of the need to create many things, retrieve and compare it, and (as turned out after trial and error) also having unique data and cleaning them afterwards to ensure correct testing of other methods. For testing endpoints, it was also simply quite time consuming.
- **Problem**: @RequestBody
    - I had trouble figuring out how to do it correctly in the controller, and as a workaround made everything accept @RequestParam, since most of the objects only have 1-2 parameters that are needed. However, I can see that it is not a clean way of doing it at the end of the day.
- **Problem** : Pagination implementation.
    - The focus was first and foremost on implementing basic functionality, and pagination was overlooked.
    - **Resolution**: Implemented a manual toPage() method to paginate lists returned from repositories as a quick fix, but should have used Pageable in JPA instead.
- **Problem** : Lack of time for advanced features (e.g., GraphQL/WebSocket).
    - Focused on meeting core requirements first.

Initial planning vs Actual planning

- Initial: Allocate equal time for functionality, testing, and documentation, start coding as early as possible.
- Actual: More time spent on testing and implementation due to lack of familiarity with the framework. Most time spent on reflection of the endpoints and application structure, once it was figured out, the rest went smoothly enough. Much time spent on Authentication part.

Because of the determination to implement basic functionality first and foremost, several features were cut altogether, such as:

- Editing discussions
- The user being able to delete their own comment/discussion (as in old forums)

Summary

- Achievements:
    - Implemented core functionality: CRUD for discussions, comments, and categories.
    - User/admin role management secured endpoints.
    - Everything tested (services with Unit tests, and Endpoints with Postman)
    - Pagination implemented (manual workaround).
- Limitations:
    - Basic pagination lacks advanced features.
    - Missing optional features like GraphQL/WebSocket.
- Evolution:
    - Future enhancements: Native pagination, use of @RequestBody, Swagger/OpenAPI for documentation.
    - Better handling of endpoint Responses, in specifics, to return more meaningful errors instead of a null body

Guide

No specifics for installation should be needed, cloning and launching spring boot application should be enough.

It is best to create an admin user right away for testing, by choosing name and password and setting role to “ROLE_ADMIN”.