La Frappe

Evann DREUMONT, Antoine REY,
Quentin POINTEAU, Alexandre VANICOTTE-HOCHMAN

Juin 2025

1 Introduction

La Frappe est un projet historique maintenu par l'association net7 depuis de nombreuses années. Ce service, à destination des étudiants de l'INP (ENSEEIHT, ENSAT, ENSIACET, etc.), permet de partager entre eux des documents de révision de façon ordonnée, sécurisée via un portail étudiant. De nombreux problèmes de conception du portail actuel tels que le cookie d'authentification qui disparaît lors de la navigation, ou les fonctionnalités de recherche et de filtrage qui ne fonctionnent pas nous ont poussés à reprendre le projet. Nous avons donc choisi de reprendre le projet, pour le réadapter aux besoins modernes, notamment pour transformer ce service qui était entremêlé dans le code d'autres projets, afin d'en faire un micro-service dépendant simplement d'une authentification centralisée. Finalement, ce service intégralement réécrit avec des technologies étudiées par les étudiants de l'ENSEEIHT permettra aux futures générations d'entretenir et de faire évoluer le projet.
Repo GitLab : `https://git.inpt.fr/inp-net/frappe`

2 Base de données

Vous trouverez ci-dessous le schéma de la base de données avec les liens entre les entity beans. Lorsqu'une seule des cardinalités est indiqué, cela veut dire que la relation est unidirectionnelle.

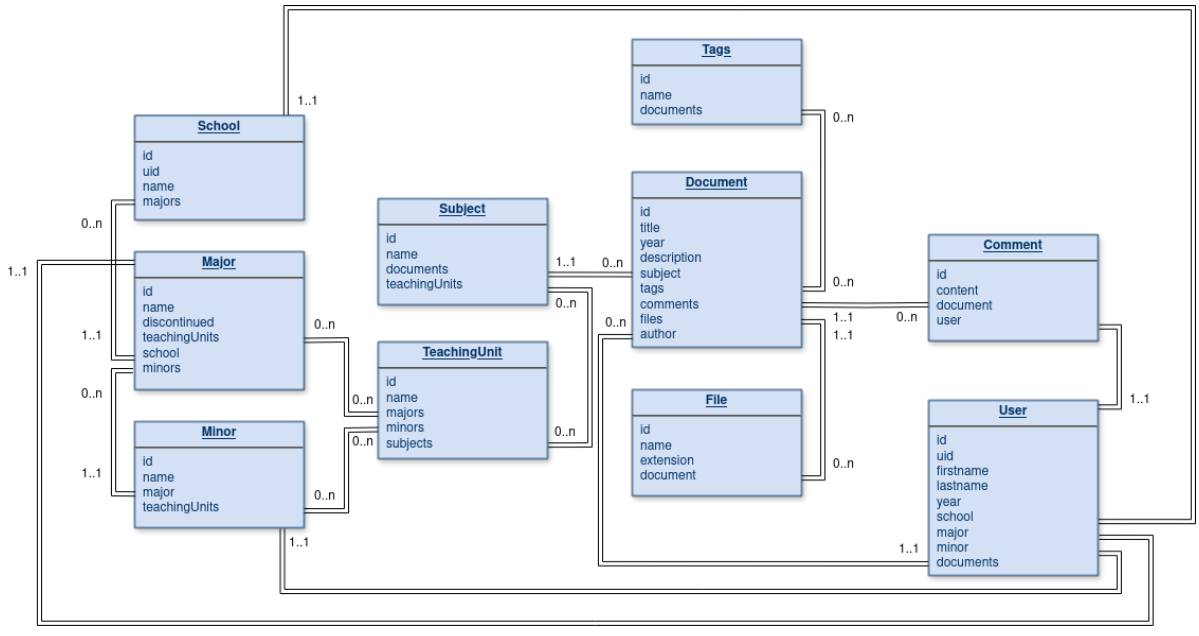

Figure 1: Base de données de La Frappe

3 Authentification

L'authentification de l'utilisateur s'effectue en trois étapes principales, impliquant plusieurs redirections entre les différentes couches de l'architecture (Frontend, API, SSO). Nous utilisons l'instance Authentik d'INP-net, ce qui nous permet de récupérer les données de churros et donc l'école, la majeur, l'année, etc...

Lorsqu'un utilisateur se connecte sur La Frappe, il est redirigé automatiquement vers l'API, qui est responsable de la mise en œuvre du protocole OAuth.

L'API déclenche le flow OAuth standard en redirigeant l'utilisateur vers l'instance Authentik d'INP-net. Le flow suit le déroulement suivant :

- L'utilisateur est redirigé vers l'interface d'Authentik pour s'authentifier.
- Une fois l'authentification réussie, Authentik redirige l'utilisateur vers une URL de callback spécifiée par l'API.
- L'API utilise cette redirection pour récupérer le token OAuth (access token, et éventuellement refresh token).

À partir de ce token, l'API identifie l'utilisateur, vérifie son identité. Et crée un token JWT. Ce token sera renvoyé à l'utilisateur via le callback de l'application.

4 Principaux choix techniques

4.1 Tech stack

4.1.1 Frontend

On utilise SvelteKit. De plus, on utilise une spécification Open-API pour être type safe. Cette documentation est générée automatiquement via un script à partir des annotations Springdoc.

4.1.2 Backend

Pour ce qui est du code côté serveur, on utilise SpringBoot, JPA repository et Spring-Doc. On utilise aussi Swagger qui nous met à disposition une interface graphique nous permettant de tester toutes les routes de l'API, toujours à partir des annotations Springdoc.

4.2 Data Transfer Object (DTO)

On utilise des DTOs pour pouvoir construire les corps des requêtes. Cela permet de centraliser les informations nécessaires pour créer ou modifier les entity beans.

4.3 Mappers

Les mappers permettent de créer et modifier les entity beans dynamiquement directement via les DTOs. Ils sont aussi très utile pour modifier qu'une partie des champs d'une entity bean en ignorant la mise à jour des champs remplis à `null`.

4.4 Spécifications pour le filtrage

`https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html`

La spécification permet de faire des requêtes via `JpaSpecificationExecutor` (que nos `JPARepository` héritent), cela permet de faire des requêtes avec des contraintes. Par exemple si on veut tous les documents qui ont pour majeure "Sciences du Numérique" et pour Mineure "ASR".

4.5 Pagination

La pagination est réalisé directement par les `JPARepository`. A terme, on aura beaucoup de documents donc pour éviter que les requêtes soit trop longues, on retourne qu'une partie des données (via un `PAgeable`). Sur le front, on utilise le package `infinite-scroll` de Svelte, qui permet que lorsqu'on arrive en bas de la page, refait une requête pour compléter les données.

5 CI/CD

Pour le backend, nous avons réalisé des tests sur les contrôleurs des différentes ressources. Nous avons rajouté de la CI qui permet de lancer un pipeline qui exécute les tests à chaque `push` sur le dépôt git et à chaque `merge` sur la branche `master`. Cela permet d'éviter de faire des modifications qui par la suite pourraient induire des régressions.

Pour le frontend, grâce au annotations Springdoc réalisé dans le backend, on génère une spécification Open-API, qui permet de faire du typage fort sur les variables. Cela permet en CI, de tester si les types sont corrects. On teste aussi si les fichiers sont correctement formatés et si ils sont correctement *lint*.

6 Déploiement

Pour finaliser le projet, nous avons décidé de déployer le projet dans un environnement de production. Pour ce faire, nous reposons sur des images docker qui sont build par la CI. Ces images sont ensuite déployées sur le cluster Kubernetes de net7. Le déploiement repose sur du gitops (fluxcd) afin de versionner le déploiement et d'assurer un état stable du cluster. Ce déploiement permet scalabilité et haute disponibilité.