Ftreel

Un projet de .NET

Jordan Elie, Sébastien Guerrier, Léo Lovicourt, Alexandre Merre, Mathieu Vallée

Janvier 2024

Table des matières

1 Introduction

L'application réalisée, `Ftreel`, est un gestionnaire de documents. Celle-ci est composée d'une `API REST` en `.NET 7`, d'un client en `React.JS`, d'une base de données utilisant le `SGBD SQL Server` et d'un serveur `SMTP`.

Ftreel met à disposition un panel d'administration dédié aux administrateurs de l'application : il leur permet de gérer les différentes catégories et sous-catégories de documents, et de mettre en ligne des documents en les classant dans les catégories existantes. Les administrateurs ont également la possibilité, depuis ce panel, de valider ou non un document proposé par un utilisateur.

Les utilisateurs de Ftreel peuvent ensuite visualiser les documents mis à disposition dans leurs catégories et sous-catégories respectives. Il leur est également possible de rechercher des documents de manière spécifique, en les triant par catégorie, titre, description et auteur du document.

Tout utilisateur peut télécharger un document mis en ligne, mais aussi télécharger tous les documents d'une catégorie ou d'une sous-catégorie en une seule fois. Il peut aussi proposer l'ajout d'un document à une sous-catégorie existante, et consulter visualiser ses propositions de documents en attente de validation d'un administrateur.

Un utilisateur a également la possibilité de s'abonner à une catégorie ou à une sous-catégorie afin d'être notifié par mail lorsqu'un document est ajouté à cette catégorie/sous-catégorie. Le modèle de ce mail est modifiable par les administrateurs depuis le panel d'administration. Un utilisateur peut aussi aimer un document, et de savoir combien de personnes ont aimé ce document.

Pour finir, les textes statiques de l'application sont disponibles en anglais et en français.

2 Répartition du travail

La réalisation de l'API ainsi que la mise en place du schéma de base de données a été majoritairement réalisée par Alexandre et Mathieu. Le client web, lui, a été réalisé par Jordan, Sébastien, et Léo.

Cependant, bien que nous nous soyons séparés en deux équipes (back-end et front-end), nous avons veillés à communiquer régulièrement sur nos avancées afin de pouvoir intégrer facilement les changements de l'autre équipe.

3 Démarrage du projet

Afin de démarrer notre projet, nous pouvons faire les commandes suivantes :
Pour lancer la base de données et le serveur SMTP :
`docker compose up` depuis le dossier de l'API.
Pour lancer l'API :
`dotnet ef database update` pour lancer les migrations. `dotnet run` pour lancer le back.
Pour lancer le client depuis le dossier du client :
`npm install npm run dev`
Il y a 2 utilisateurs par défaut dans notre application :
— admin@ftreel.com qui a pour mot de passe admin.

— user@ftreel.com qui a pour mot de passe user.

4 Travail réalisé

4.1 API

4.1.1 Architecture globale

L'API du projet s'articule principalement autour de 3 dossiers, comme le veut l'architecture d'un projet Spring Boot.

— `Controllers` : Contient toutes les routes de l'API.
— `Services` : Contient la plupart du code avec une logique métier.
— `Entities` : Contient nos objets à persister en base de données.

A cela se rajoutent d'autres dossiers :

— `DATA` : Contient la classe AppDBContext héritant de DbContext et servant à faire des requêtes à la base de données, ainsi qu'à configurer les relations.
— `Migrations` : Contient les migrations de base de données du projet.
— `Dto` : Contient les objets servant à recevoir et envoyer des données. Nous nous servons donc des objets contenus dans ce package pour échanger des données avec le client.
— `Constants` : Contient les constantes utilisées dans le projet, par exemple les rôles utilisateurs de l'API.
— `Exceptions` : Contient quelques exceptions personnalisées, comme pour le système de stockage.
— `Utils` : Ce package ne contient que un module pour gérer les mots de passe et leur hachage.

Le dossier Controllers est le dossier gérant les routes de l'API. Toutes les classes sont suffixées par "Controller". Il y a cinq controllers :

— `AuthenticationController` : Ce controller gère toutes les routes liées à l'authentification. Il s'agit donc du login, du logout, et de la récupération de l'utilisateur actuellement connecté.

— `UserController` : Ce controller a un CRUD (Create, Read, Update, Delete) de toutes les routes liées aux utilisateurs.

— `DocumentController` : Ce controller a un CRUD de toutes les routes liées aux documents. Il contient également la gestion de la validation des documents et des likes.

— `CategoryController` : Ce controller contient un CRUD de toutes les routes liées aux catégories des documents. Il contient également la gestion des abonnements.

— `WeatherForecastController`

Les controllers en eux-mêmes gèrent uniquement les réponses. Par exemple, un controller va renvoyer un code d'erreur 200 ou 204 si la requête est exécutée avec succès, ou alors une 401 si l'utilisateur n'est pas connecté, 403 si l'utilisateur n'a pas accès à la route, 404 si l'objet demandé n'existe pas ou alors 400 pour tout autre erreur.

Ces controllers sont liés aux différents services, qui sont :
— `AuthenticationService` : Ce service contient les méthodes pour pouvoir connecter un utilisateur, pour avoir l'utilisateur connecté, et sert aussi dans les autres services et controllers.
— `IUserService` : Contient un CRUD afin d'interagir avec les différents utilisateurs de l'application. En plus de cela, ce service contient des méthodes privées pour vérifier si l'email est bien formé.

— `IDocumentService` : Contient un CRUD pour interagir avec les documents. Les méthodes interagissent avec la base de données mais aussi avec le système de stockage de fichiers.

— `IStorageService` est une interface qui définit le système de stockage, pouvant récupérer un document à partir d'un nom de fichier, le supprimer, etc ...

— `ICategoryService` Contient un CRUD pour interagir avec les catégories en base de données.

— `IMailService` sert à envoyer des mails à un utilisateur.

Il y a trois entités dans notre projet :
— `User` est l'entité représentant un utilisateur. Celle-ci contient un email, un mot de passe et une liste de rôles. Le mot de passe est chiffré avant sa persistance en base de données, avec l'algorithme BCrypt.

— `Document` est l'entité modélisant un document. Elle contient le titre du document, une description du document, la validité du document, et le type MIME de contenu du document. Elle contient également un attribut qui n'est pas sauvegardé en base de données (avec l'annotation `[NotMapped]`) : il s'agit de la base 64 du document. Les documents sont stockées en fichiers directement dans le répertoire `/upload` à la racine de l'API, ce qui permet de ne pas avoir une base de données trop lourde, et de prendre moins de temps à charger ces documents. Les documents contiennent aussi une catégorie, un auteur (utilisateur) et une liste d'utilisateurs qui ont aimé le document.

— `Category` est l'entité correspondant à une catégorie. Elle contient un nom, la catégorie parente si elle existe, les catégories enfantes, les documents enfants et une liste d'utilisateurs qui suivent la catégorie.

4.1.2 URLs et DTOs

Les différentes URLs et DTOs de l'API sont définies dans le Google Doc accessible en cliquant sur le lien suivant. Lorsque l'API tourne, vous pouvez également consulter le swagger pour jouer avec.

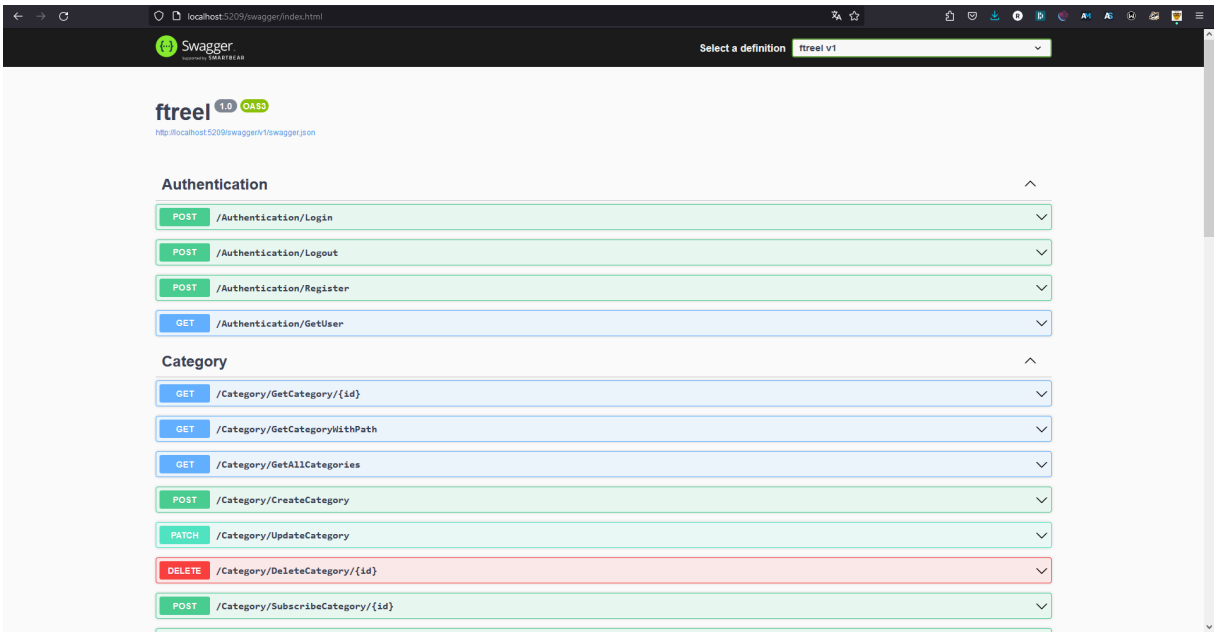

FIGURE 1 – Swagger

4.1.3 Gestion des logs applicatifs

Nous ne savions pas tellement au début de quoi il s'agissait comme bonus, mais nous avons essayé de faire cette partie en faisant des logs de tout ce qui se passe dans l'application. En effet, pour chaque requête effectuée à l'API, un message est affiché par l'API pour savoir ce qui a été fait dans l'API.

Nous avons donc désactivé les messages affichés par les requêtes de la base de données pour pouvoir voir les logs.

Les logs sont réalisés grâce à l'interface ILogger de C#.

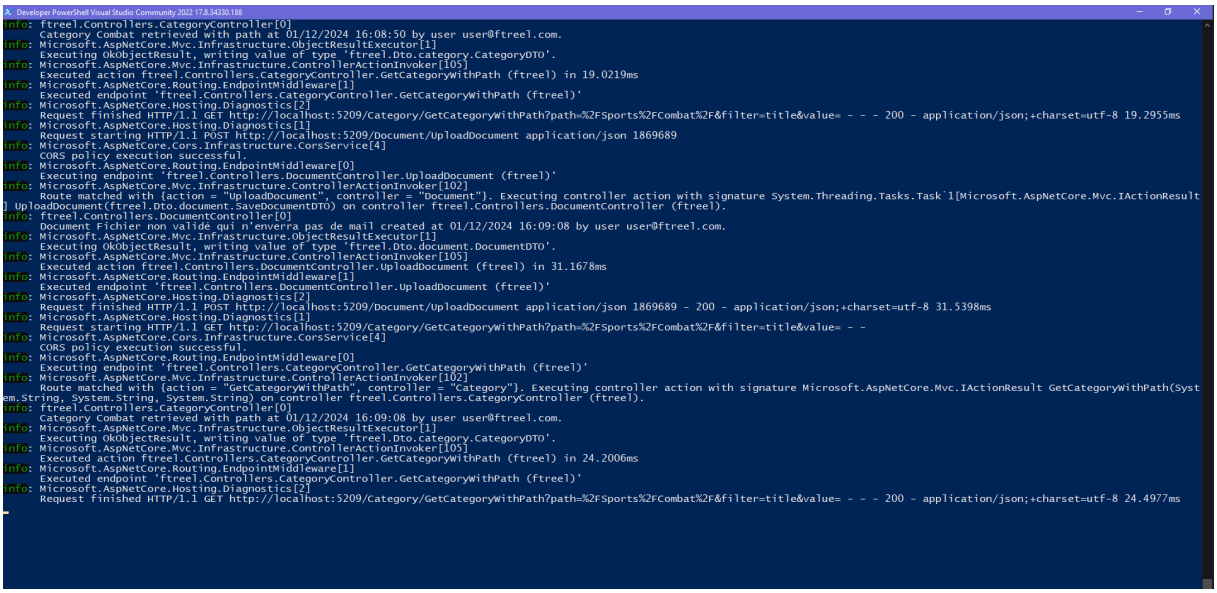

FIGURE 2 – Logs

4.2 Base de données

4.2.1 Schéma

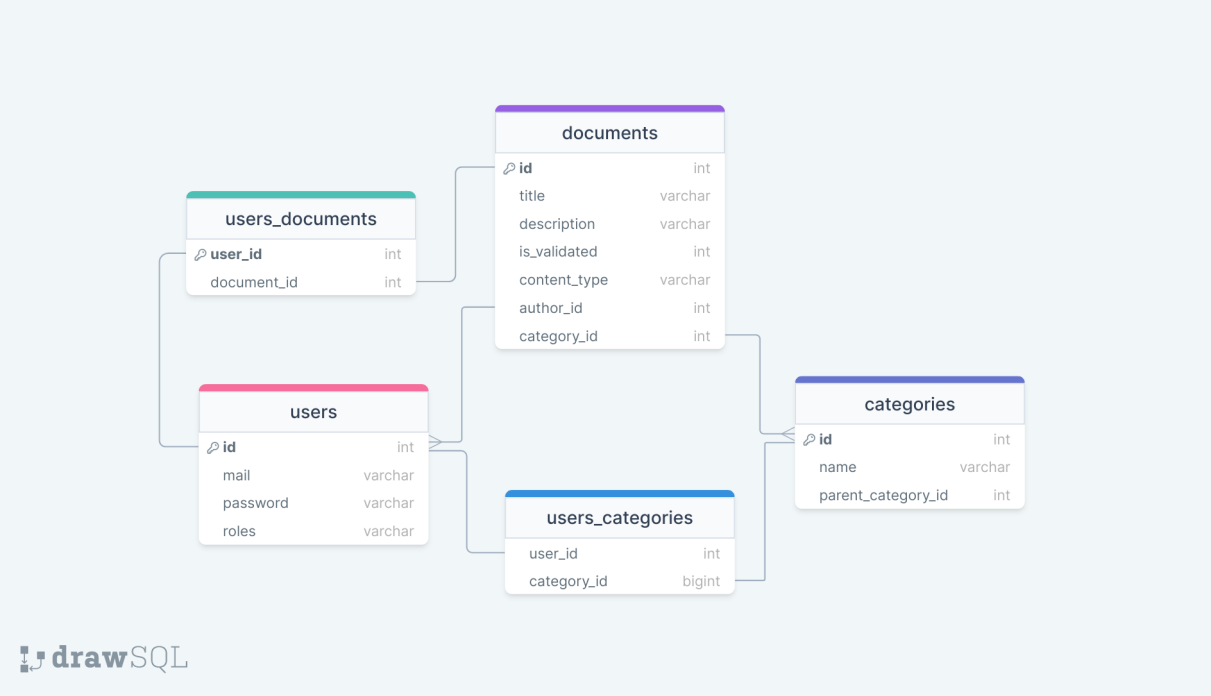

FIGURE 3 – Schéma relationnel de la base de données

4.2.2 Explication

Les utilisateurs (table `users`) ont une adresse mail, un mot de passe chiffré en base de données et une liste de rôles, séparés par des ";".

Les documents (table `documents`) ont un titre, une description, un état de validation, et un type de contenu correspondant au type MIME du document.

Les catégories (table `categories`) ont un nom.

Il y a plusieurs relations entre les tables :
- Les utilisateurs ont une relation Many-to-One avec les documents, car ils peuvent être auteurs de plusieurs documents.
- Les utilisateurs ont une relation Many-to-Many avec les catégories car ils peuvent suivre plusieurs catégories, et une catégorie peut avoir plusieurs abonnés. La table de jointure est `users_categories`.
- Les catégories ont une relation Many-to-One avec les documents, une catégorie peut avoir plusieurs documents.

— Les catégories ont une relation Many-to-One avec elles-mêmes car elle peut avoir plusieurs sous catégories.
— Les utilisateurs ont une relation Many-to-Many avec les documents pour gérer les likes. En effet, un utilisateur peut aimer plusieurs documents, et un document peut être aimé par plusieurs utilisateurs.

4.3 Serveur SMTP

Le serveur SMTP a été mis en place à l'aide de l'image Docker `dockage/mailcatcher` utilisée dans le `Docker compose` fourni.

Nous envoyons donc des mails à l'aide du service `IMailService` et de son implémentation `MailService` à chaque fois qu'un document est validé et qu'il appartient à une catégorie. L'adresse mail qui envoie le mail est `ftreel.system@ftreel.com`.

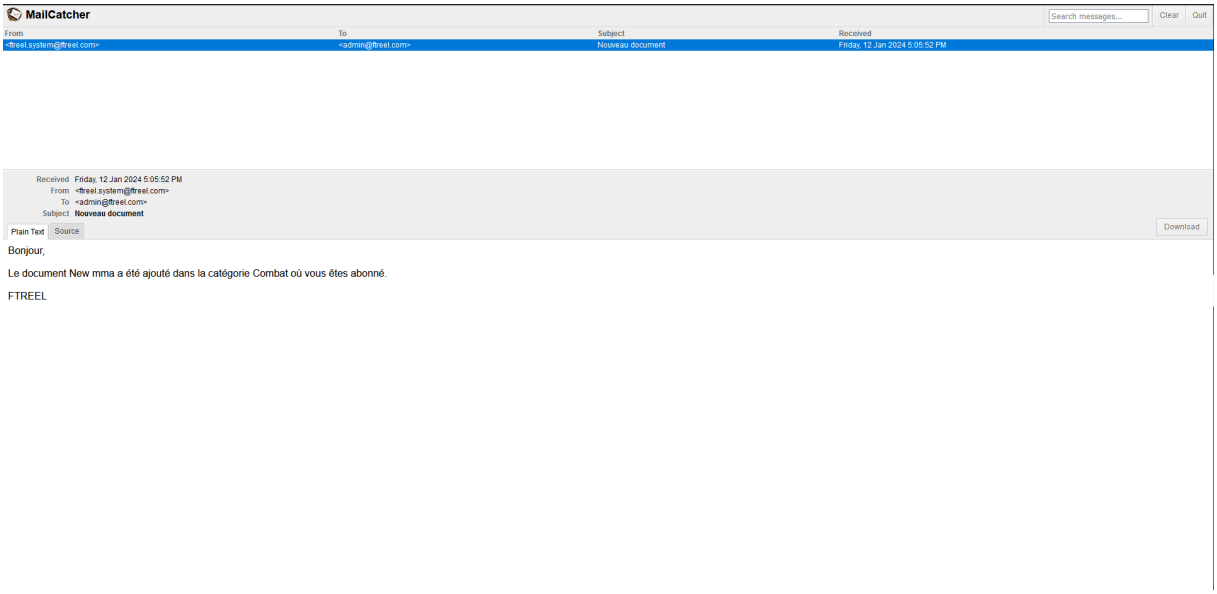

FIGURE 4 – Mail catcher

4.4 Client web

La partie client a été réalisée en utilisant la librairie React, accompagné du langage TypeScript. Le choix de cette librairie a été fait dans une démarche d'apprentissage de la technologie, ainsi qu'une préférence dans la manière de créer le template grâce à la syntaxe JSX.

L'interface est composé des pages suivantes :

— L'accueil : Cette page permet d'accéder au gestionnaire de document, ainsi que de se connecter et s'inscrire.
— Le gestionnaire de document : Cette page permet de naviguer entre les documents, ainsi que de les manipuler (création, modification, suppression...)

— Les documents suivis : Sur cette page, on peut visualiser les différentes catégories suivies par l'utilisateur connecté.
— La gestion des utilisateur : Cette page uniquement accessible par les administrateurs permet de gérer les utilisateurs de l'application (création, suppression...). L'utilité principale d'une telle page est de pouvoir créer des administrateurs.
— La page de validation : Sur celle-ci, on retrouve les documents qui n'ont pas encore été validés, un administrateur peut donc venir valider les différents documents présents dans la liste.

En plus de toutes ces pages, le client implémente de la traduction en anglais, ainsi qu'une partie responsive afin de pouvoir visiter l'application depuis un smartphone.

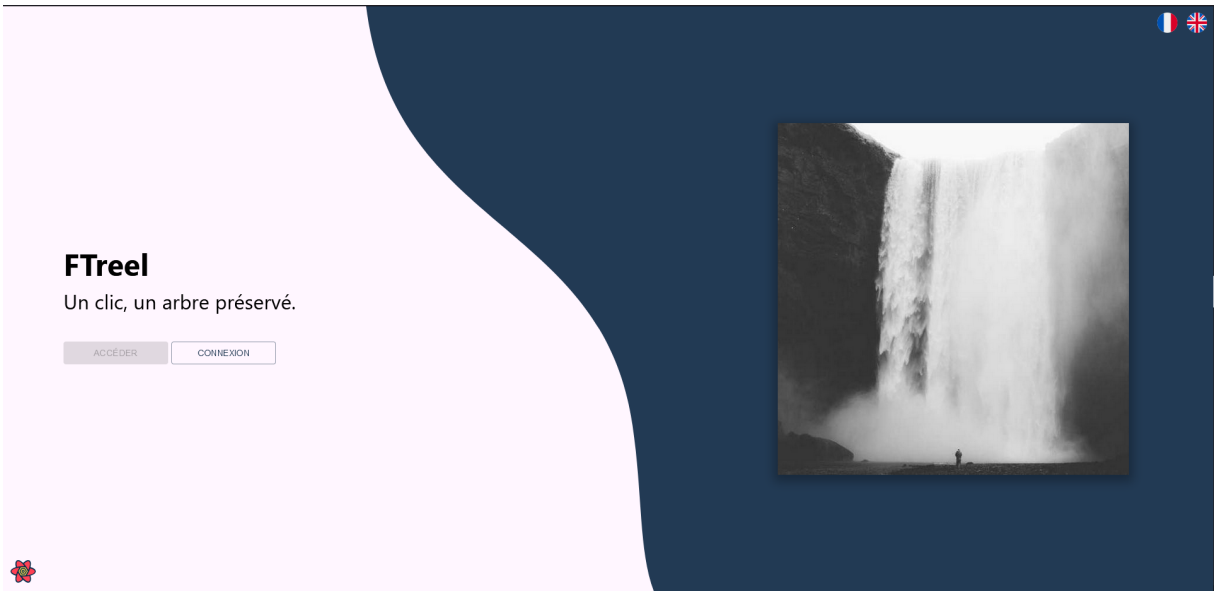

FIGURE 5 – Vue du client

FIGURE 6 – Vue du client

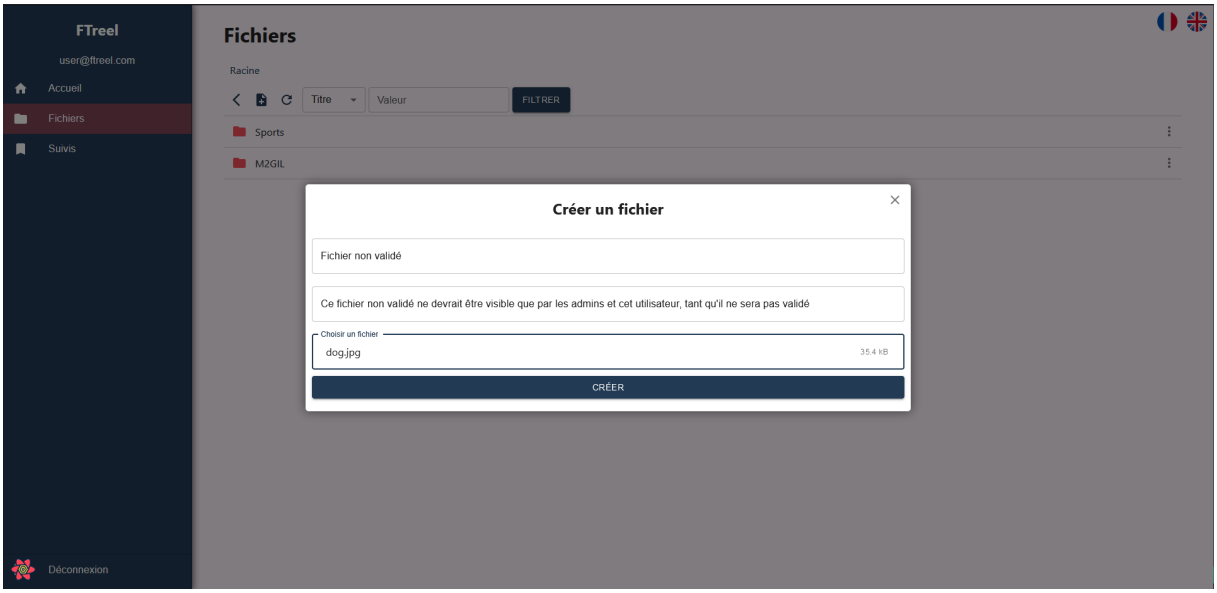

FIGURE 7 – Vue du client

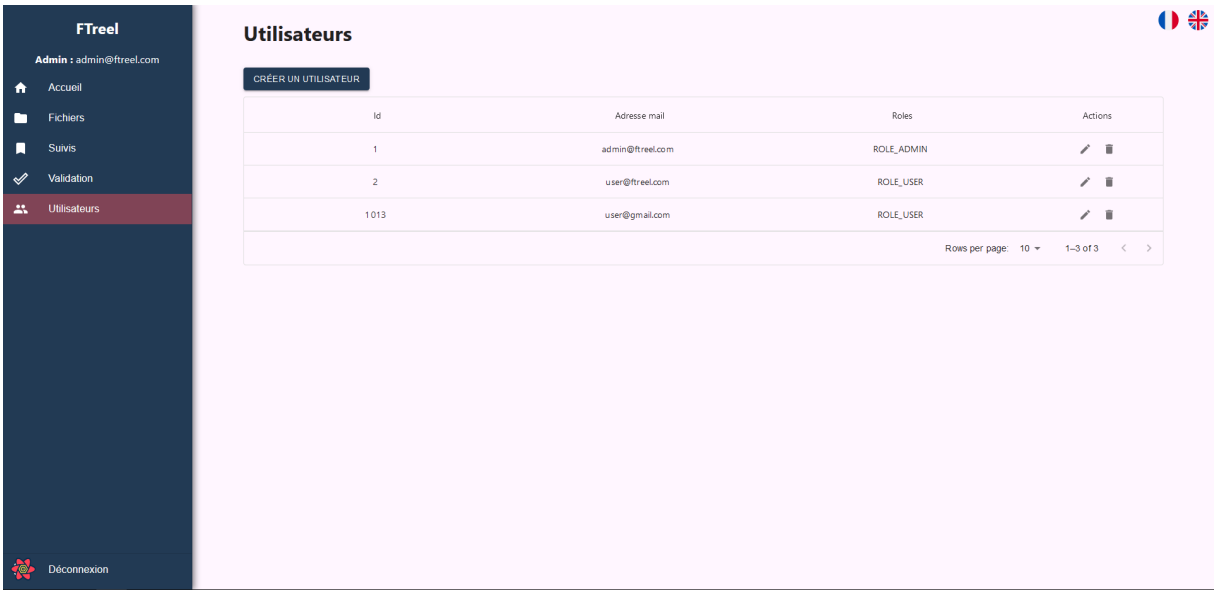

FIGURE 8 – Vue du client

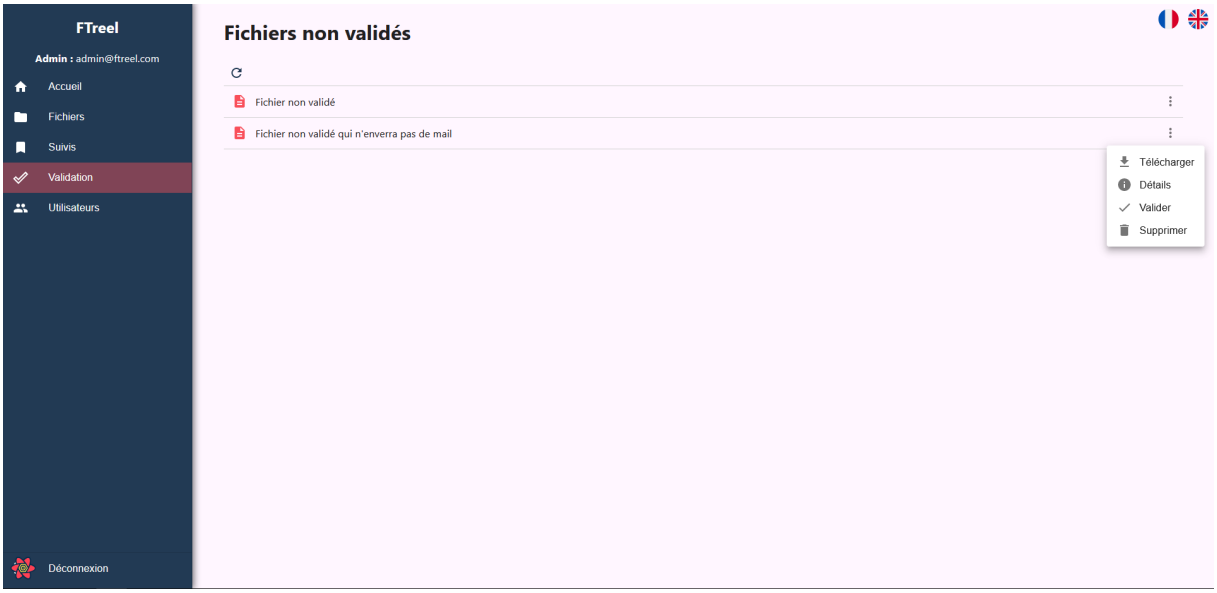

FIGURE 9 – Vue du client

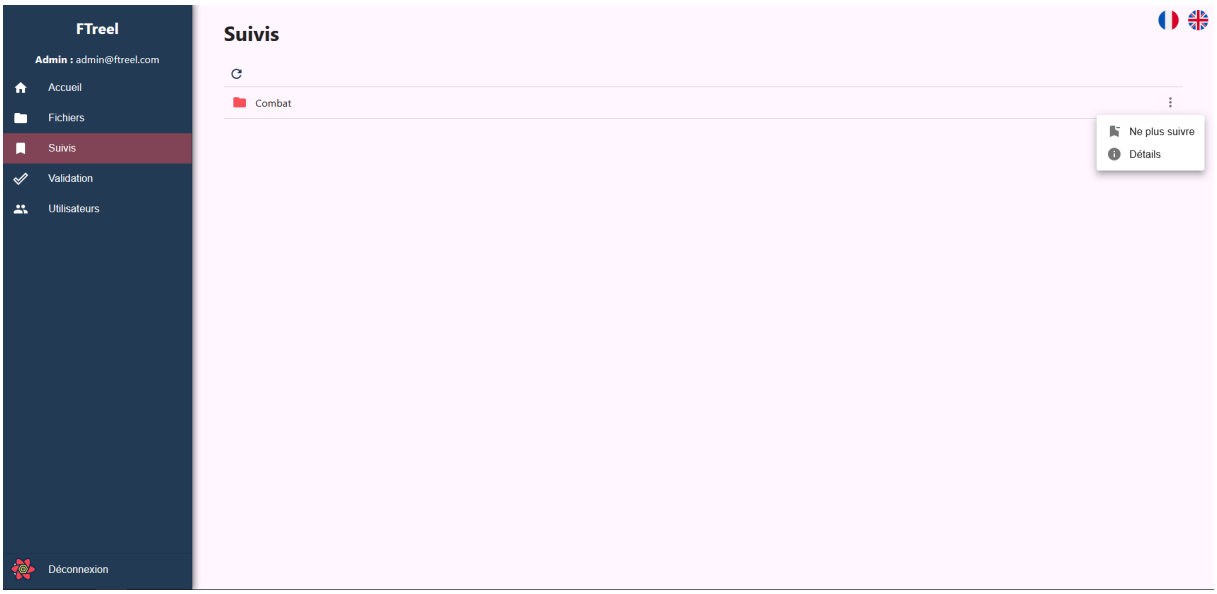

FIGURE 10 – Vue du client

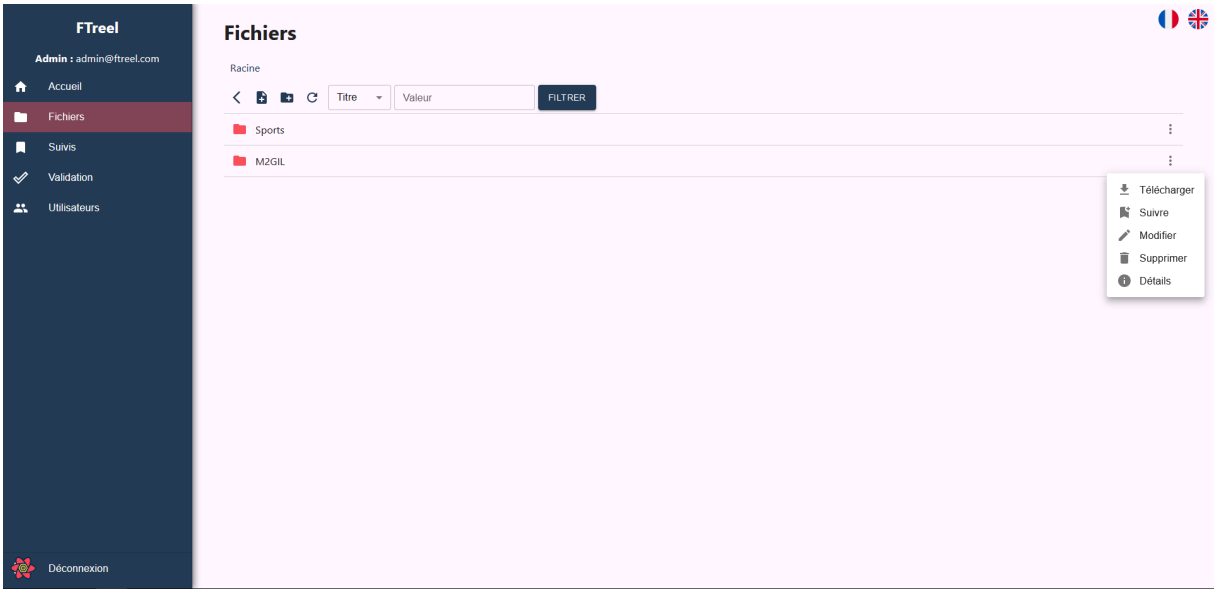

FIGURE 11 – Vue du client

5 Difficultés rencontrées

5.1 API

La plupart des difficultés rencontrées dans l'API sont des incompréhensions ou des choses qui ne nous ont pas semblé intuitives avec le framework .NET. En effet, ayant eu un projet similaire dans une autre matière, nous n'avons pas rencontré de réelle difficulté lors de la conception de l'envoi de documents ou encore pour le système de catégorie.

Nous n'avons pas non plus eu trop de mal avec le C# étant donné que c'est assez similaire au Java.

5.1.1 Listes en base de données

La première difficulté a été de faire une liste de chaîne de caractère pour la stocker en base de données. En effet, cela n'est pas géré nativement par le framework .NET. Nous avons donc dû faire en sorte que les objets en base de données soient séparés par un ';'. Ainsi, pour le stockage des rôles des utilisateurs, nous pouvons avoir ceci stocké en base de données :

`ROLE_USER;ROLE_ADMIN`

5.1.2 Redirection lorsque non connecté ou non autorisé

Si on mettait l'annotation `[Authorize]` sur une route et que l'on était ni connecté, ni autorisé, au lieu d'avoir une erreur 401 ou 403, nous étions redirigés, ce qui donnait une erreur 404.

Nous avons eu beaucoup de mal à identifier ce comportement étrange, et nous nous sommes rendu compte qu'il fallait régler les redirections dans notre fichier `Program.cs`.

5.2 Client web

La principale difficulté dans la réalisation du client web était l'appropriation de la librairie React. En effet, malgré sa simplicité en apparence, dans le cadre d'une application plus complexe, celle-ci peut devenir plus ardue à maîtriser.

La deuxième difficulté fut le responsive, créer une sidebar capable de s'adapter aux différentes tailles d'écran en React était une première pour nous, et ne fut pas une tâche des plus évidentes.

Enfin, nous avons eu des difficultés concernant l'actualisation de l'application lorsqu'un utilisateur se déconnecte. Le layout ne s'actualisait pas correctement et laissait apparaître des éléments d'un utilisateur connecté. Nous avons pu résoudre cela grâce à un layout global qui se charge de la gestion de l'actualisation et des redirections lors du changement d'état d'un utilisateur.

5.3 Fonctionnalités bonus

Nous avons permis une hiérarchie infinie de sous-catégories (pouvoir créer une sous-catégorie d'une sous-catégorie et lui attribuer des documents) comme il l'était suggéré dans le sujet. Nous avons également mis en place la gestion des likes, pour pouvoir liker, retirer son like sur un fichier et également voir son nombre de likes.

6 Axes d'amélioration

6.1 API

6.1.1 Envoi des documents

Pour l'envoi des documents, nous le faisons en JSON et nous envoyons le contenu en base 64. Cependant, nous nous sommes rendu compte que utiliser le type d'envoie `multipart/form-data` pouvait être plus optimisé pour l'envoi de document.

6.1.2 Vérifications des données envoyées

Dans l'API, nous vérifions déjà beaucoup de choses lorsque l'on crée une catégorie, un document, à la modification, etc. Cependant, tout n'est peut être pas vérifié comme il le faut, nous avons cependant fait en sorte que cela marche avec le client sans bug.

7 Conclusion

Pour conclure, l'application est fonctionnelle et répond aux exigences formulées dans le cahier des charges. La réalisation de celle-ci nous a permis d'apprendre à utiliser le framework .NET conjointement avec un client web en React.JS, notamment à travers un système d'authentification, une gestion des rôles, et la persistance de documents.