Rapport de projet – UJM Docs

AKIL Mourtaza, BAYAZID Hany, RIEU Valentin, ROMAIN Bruno

L3 Informatique | Développement Web II

10 mai 2023

Table des matières

1 Introduction

Dans le cadre du projet de développement web II, nous avons décidé de réaliser une application de traitement de texte collaborative dans le style d'un éditeur simple comme Notepad mais offrant la possibilité de travailler simultanément sur un même document à plusieurs.

Pour cela, il nous a été conseillé de développer un serveur central, qui accepterait des connexions de clients, les clients étant représentés par une application lourde et/ou par une application légère.

Notons que l'application doit s'installer sur l'ordinateur, et l'application légère ne nécessite pas d'installation, elle est juste une interface liée au serveur, accessible grâce à un navigateur web.

L'application lourde disposerait d'un mode hors-ligne, qui prend la main lors de la perte de connexion internet, et qui fait le nécessaire une fois celle-ci retrouvée, ce qui permet d'avoir un éditeur fonctionnant localement, où la notion de collaboration est mise de côté, accentuant ainsi la polyvalence de notre projet.

Pour cela nous allons dans un premier temps présenter la structure de notre projet, ensuite, nous allons détailler plus profondément ses différents modules.

2 Conception de la plateforme

Nous utilisons une base de données pour stocker les informations relatives aux documents et aux utilisateurs, dont voici le schéma de données :

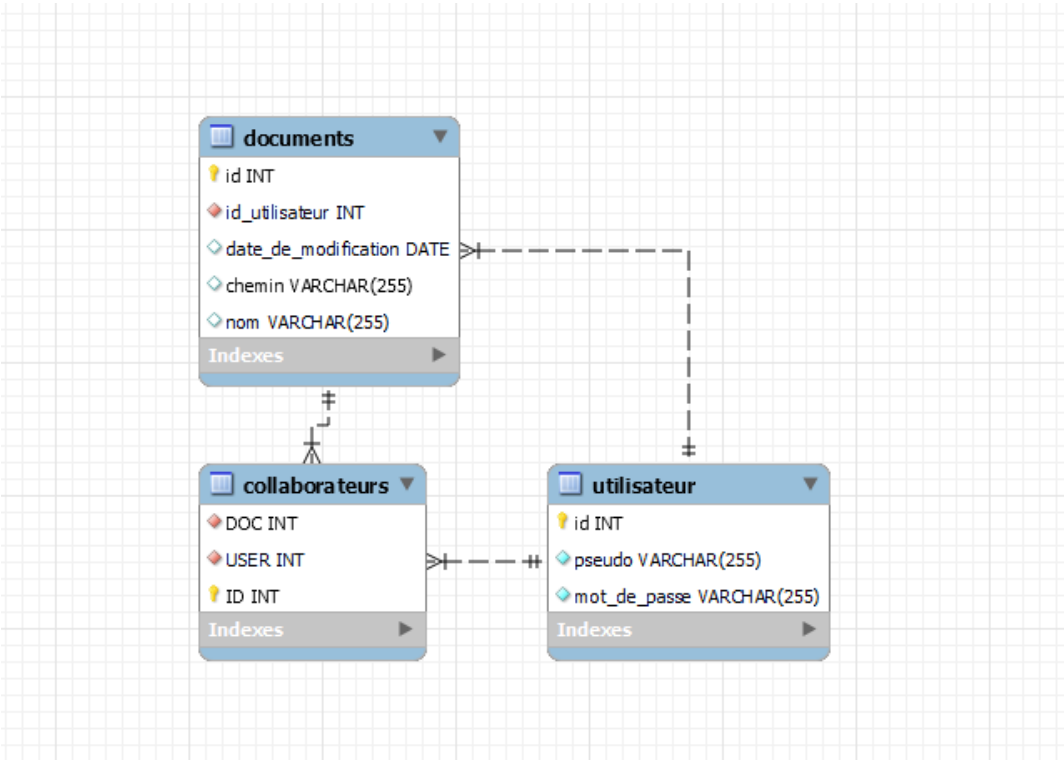

Figure 1: Modèle entité-association de la base de données

Nous avons deux entités, des *documents* et des *utilisateurs*. Un document est relié à un utilisateur particulier, qui est son créateur. Comme nous voulions implémenter un certain niveau de privatisation des documents, nous avons une relation entre les documents et les utilisateurs, qui est la collaboration : un **ID** de document peut être associé à un ou plusieurs utilisateurs, qui correspondent aux collaborateurs travaillant sur le document.

Est décrite dans le schéma suivant l'utilisation prévue de nos applications :

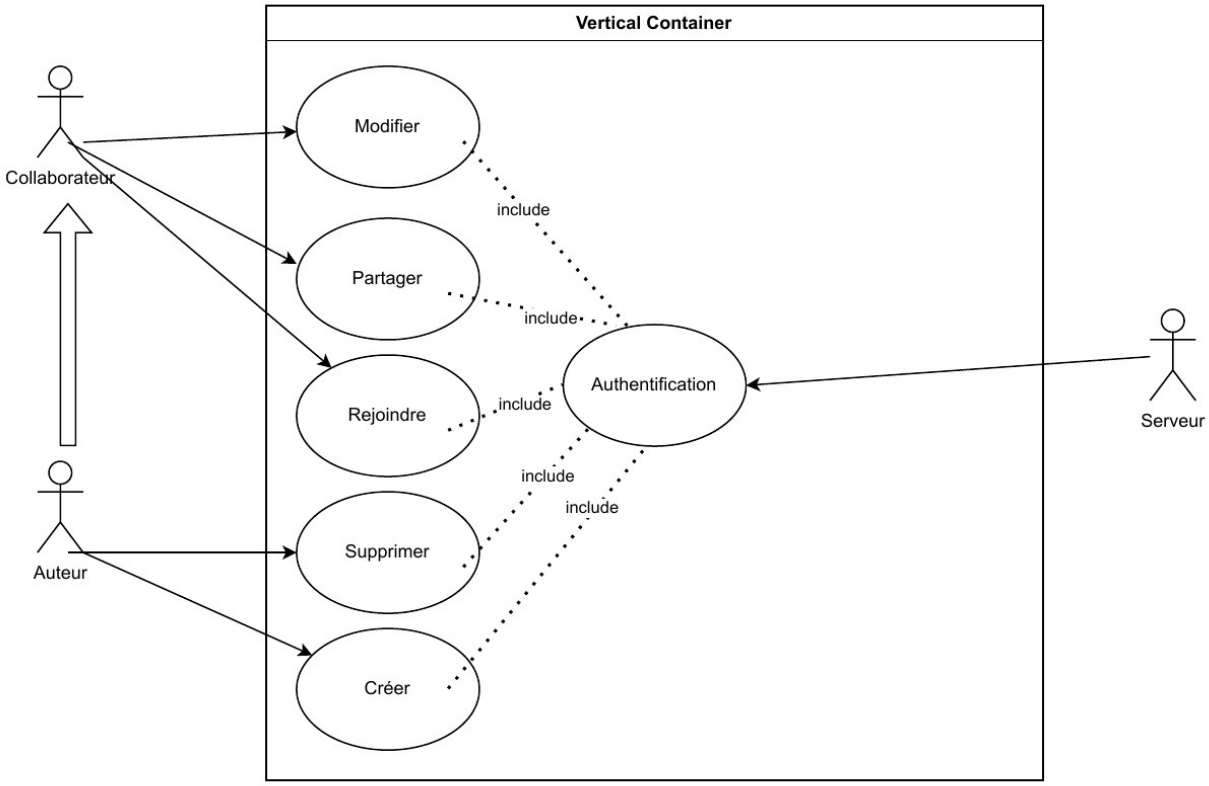

Figure 2: Diagramme des cas d'utilisation de la plateforme

3 Serveur

Le serveur a pour but d'instancier des connexions pour chaque client, et doit synchroniser les dites connexions afin d'obtenir l'effet collaboration.

Une de nos premières idées a été d'implémenter un serveur **Spring Boot**, mais la communication via des requêtes ***HTTP*** n'ayant pas été possible par le sujet du projet et des difficultés à son instanciation, elle fût abandonnée assez vite, malgré sa supériorité sur les communications. En effet, il suffisait de construire un ***rest API,*** et seuls les clients demanderaient au serveur, à intervalle de temps régulier, différentes requêtes : ***GET*** pour lire les infos, et ***POST*** et/ou ***DELETE*** pour les envoyer.

3.1 Représentation d'un document

En terme de représentation des données d'un document, nous étions initialement partis sur une représentation par une liste chaînée de mots, car sur le papier, cette structure permet de simplifier la logique pour insérer des données de manière multiple à plusieurs endroits différents. En effet, dans notre cas, comme un mot aurait été défini dans la liste en fonction de son prédécesseur et son successeur, la gestion des indices des mots dans le document aurait pu être simplifiée, et nous pensions pouvoir implémenter des fonctions telles que la prédiction de mots et l'autocomplétion.

Or, par manque de temps et à cause d'autres obligations, ce système fût abandonné pour repartir sur une chaîne de caractère classique.

Dans notre modèle, un document est composé de pages, et le contenu de chaque page est accessible uniquement par les utilisateurs qui sont dessus. Ce choix fût aussi fait pour simplifier les choses, car si deux collaborateurs modifient le document sur des pages différentes, ils n'ont pas besoin de connaître les modifications apportées par les autres. Notre vision des pages est différente d'une vision classique, dans le sens où celle-ci est limitée au niveau du nombre des caractères. Dans notre modèle, nous n'avons pas de limites (si ce n'est la capacité maximum des listes en Java, plus de 2 milliards d'éléments par liste).

Pour la sauvegarde et le chargement de fichiers, nous utilisons une librairie, *fileUtils* de *apache.commons.io*, pour nous éviter d'avoir à réécrire des fonctions de lecture et d'écriture.

4 Client lourd

Notre client lourd communique avec le client à l'aide du protocole TCP, comme indiqué dans le sujet.

4.1 Étapes de communication – Client lourd ↔ Serveur

Le processus de communication se déroule de la manière suivante :

- **Authentification :** un client tente de s'authentifier en entrant son pseudo et son mot de passe sur le panneau de connexion.
- **Vérification → Choix de l'action initiale :** si le client a renseigné des identifiants corrects, un panneau lui offrira le choix entre la création d'un document ou l'ouverture d'un document déjà présent dans la base de données.
- **Initialisation de l'éditeur :** le choix de l'utilisateur sera envoyé sous forme d'un code qui permettra au serveur de retourner la réponse adéquate.
    - **Création de fichier :** un fichier vide est créé après avoir demandé le nom de celui-ci à l'utilisateur, puis est inséré dans la base de données.
    - **Ouverture d'un fichier existant :** le serveur répond en confirmant le chargement du fichier dont il a reçu le nom depuis le client, si celui-ci existe dans la base de données, puis envoie le contenu du fichier demandé qui sera chargé dans l'éditeur du client.

- **Modification de texte :** lorsque l'utilisateur effectue des modifications sur l'éditeur, un code est envoyé vers le serveur précisant le type de modification dont il s'agit. En effet, il y en a deux : l'ajout et la suppression de contenu.
  - **Ajout (ADD) :** lors d'un ajout de texte, le client envoie les informations suivantes :
    - *Identifiant de l'utilisateur*
    - *Page du document où est situé l'utilisateur*
    - *Position du curseur de l'utilisateur dans la* ***page (!document)***
    - *Offset du texte ajouté*
    - *Texte ajouté*

    Ces informations permettront au serveur de rajouter le texte au bon endroit dans la **page** sur laquelle travaille l'utilisateur et de mettre à jour les positions des curseurs des autres collaborateurs de la session.
  - **Suppresion (DEL) :** lors d'une suppression de texte, le client envoie les informations suivantes :
    - *Identifiant de l'utilisateur*
    - *Page du document où est situé l'utilisateur*
    - *Position du curseur de l'utilisateur dans la* ***page (!document)***
    - *Offset du texte supprimé*
    - *Longueur du texte supprimé*

    Similairement à la requête d'ajout, ces informations vont permettre d'appliquer les bonnes modifications au document et de faire les mises à jour adéquates sur les curseurs des collaborateurs.
- **Création / Chargement - En cours de session :** l'utilisateur doit avoir la possibilité de demander la création ou l'ouverture de nouveaux documents en cours de session. Malheureusement, nous n'avons pas pu terminer l'implémentation de cette fonctionnalité.

4.2 Communication dynamique

Ces échanges complexes requièrent la présence de nombreux threads de communication qui ont été implémentés du côté client comme du côté serveur, afin d'assurer une communication stable et un envoi de données correctes.

Cependant, de nombreuses difficultés ont été rencontrées durant l'implémentation des processus de communication, notamment une certaine incohérence lorsque l'ajout de caractère est fait d'une manière plus rapide que la durée requise par le serveur pour traiter l'ajout ou la suppression d'une manière cohérente.[1]

1 Par exemple, lorsque l'utilisateur tape beaucoup trop vite, voire même moyennement vite.

4.2.1 Client

Le client dispose de deux threads qui tournent en même temps que le principal sur lequel est lancée l'application swing.

Le premier attend en permanence des actions à appliquer depuis le serveur :

- **Modification de texte :** lorsqu'un utilisateur modifie le contenu d'un document, le serveur traite la modification et retourne un contenu prenant en compte la modification à tous les collaborateurs connectés.
- **Déplacement de curseur :** lorsqu'un utilisateur se déplace dans le document, il est nécessaire que les autres collaborateurs réceptionnent cette information. L'utilisateur envoie donc cette information au serveur qui la transfère à tous les collaborateurs.

Le second envoie en permanence des actions au serveur, également lorsqu'il n'y en a pas (cas particulier) :

- **Aucune action réalisée (NO_ACTION) :** le thread qui gère l'échange de « messages », sur une session de collaboration, du côté du serveur attend des messages de la part de tous les clients *lourds* à chaque itération (cf section 4.2.2). Il ne faut donc pas que le thread d'envoi du côté du client attende une action de l'utilisateur pour envoyer un message au serveur car dans le cas contraire, l'inaction de l'utilisateur bloquerait le serveur. On envoie donc un code **NO_ACTION** qui indique au serveur que l'utilisateur ne veut réaliser aucune action et qu'il peut « passer » au client suivant.
- **Autres actions :** à part ce dernier cas, le thread envoie des actions lorsqu'il en trouve dans une file d'attente (bloquante et chaînée) implémentée grâce à la classe ***LinkedBlockingQueue*** de la librairie ***java.util.concurrent.*** Cette file d'attente permet d'envoyer les actions dans l'ordre et de permettre une cohérence au niveau des modifications. On aura globalement les actions suivantes :
  - Modification de texte
  - Déplacement de curseur
  - Création, chargement, sauvegarde, exportation

4.2.2 Serveur

Le serveur accorde un thread à chaque session de collaboration[2]. Chaque session de collaboration est associée à un thread qui permet de communiquer avec tous les collaborateurs travaillant sur le document.

Il attend des données de la part des collaborateurs et les traite en intégralité. Il lit sur chaque **inputStream** du serveur TCP, associé à chaque collaborateur et sur lequel écrit le thread d'envoi. Ensuite, il retourne le contenu post-modifications à tous les collaborateurs en écrivant sur l'**outputStream** de chaque collaborateur sur lequel lit le thread de réception.[3]

2 Une session de collaboration tourne autour d'un document. Dès qu'un document est crée ou accédé par un utilisateur, une session de collaboration associée à ce document est initialisée.

3 Il s'agit en fait d'un système de roulement dans lequel les modifications rentrent pendant la réalisation d'un tour, puis une fois le tour effectué, le système envoie son état courant (Vulgarisation du concept).

4.3 Synchronisation

Pour être sûr de ne pas avoir d'incohérences à cause d'un « chevauchement » de threads, on a défini un système de manager qui pourrait se refléter un peu dans l'ordonnanceur d'un système d'exploitation. Le manager est implémentée par la classe ***Manager*** chez le client lourd et par la classe ***DocManager*** du côté du serveur. Tous les threads disposent d'une référence à un manager. Chez le client lourd, il n'y aura qu'un seul manager, alors que chez le serveur, il y en aura un par session de collaboration ouverte.

Les méthodes des managers sont toutes **synchronized**, et aucun des threads n'a accès à des méthodes autres que celles du manager. Il n'est donc pas possible qu'il y ait des problèmes de synchronisation ou d'incohérence. Voici deux schémas illustrant ce modèle :

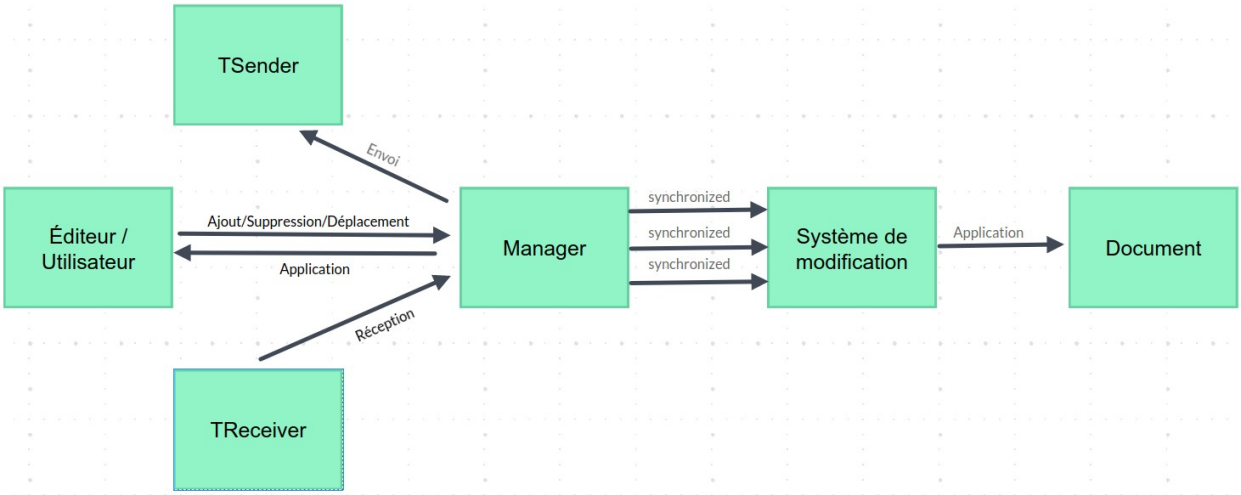

Figure 3: Schéma d'illustration du principe de synchronisation chez le client lourd

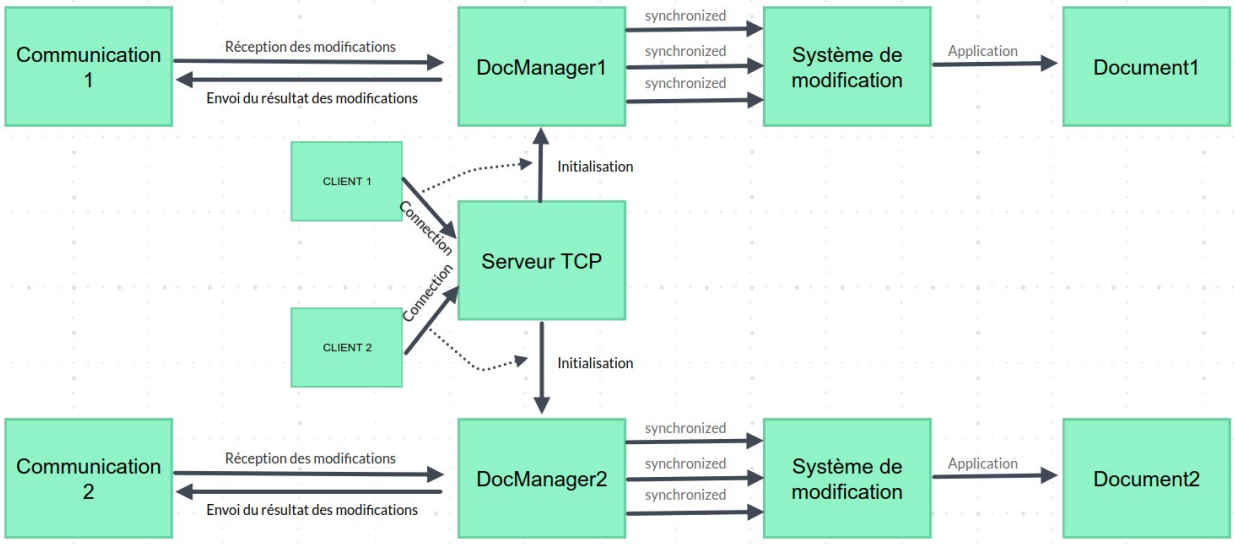

Figure 4: Schéma d'illustration du principe de synchronisation chez le serveur

5 Client léger

Nous avions prévu d'implémenter un client léger qui utilise les servlets pour l'authentification, donc la connexion et pour la possibilité de garder le client connecté au serveur après une authentification du moment que le navigateur reste ouvert, notamment à travers l'association avec les différentes portées possibles (**application, session**).

Malheureusement, à cause des nombreuses difficultés rencontrées, nous n'avons pas été capables d'implémenter un client léger qui permette une collaboration. Actuellement, il est uniquement local et statique. Nous avons toutefois pu implémenter l'interface et des fonctionnalités d'ergonomie.

Notre objectif était d'implémenter cette structure :

- Une authentification grâce aux servlets
- Une communication bidirectionnelle grâce au protocole websocket
  - Un principe de communication identique à celle du **client lourd – serveur**
  - Une synchronisation avec les modifications apportées par les collaborateurs aussi bien sur l'application web que sur l'application swing
  - Une messagerie qui vient s'intégrer à cette communication avec un formatage unique qui permette la distinction des messages par le serveur
- L'implémentation de fonctionnalités disponibles sur l'application swing
- L'implémentation de fonctionnalités extras permises par la richesse des options offertes par html, css, javascript

6 Rétrospective

6.1 AKIL Mourtaza

Je considère que le sujet du projet est particulièrement intéressant dans le sens où il est complètement différent de ce qu'on a fait jusque-là dans la formation. Je me suis notamment occupé du client lourd, ce qui m'a permis d'appréhender fortement le principe MVC, la bibliothèque swing, le protocole TCP/IP ou tout simplement le langage Java et la programmation orientée objet.

Toutefois, le projet n'ayant pas pu être finalisé, je considère qu'il y a beaucoup de notions que je n'ai pas eu l'occasion d'assimiler.

Temps de travail : j'estime mon investissement au projet à plus de 70 heures de travail.

6.2 BAYAZID Hany

Personnellement, j'ai trouvé le thème du projet enrichissant et instructif. Il m'a permis d'approfondir de nombreuses notions vues en cours, notamment la communication à travers le protocole TCP en conjonction avec les threads afin de correctement synchroniser une communication plus complexe et avancée que celle mise en œuvre durant les séances de travaux pratiques.

De plus, la réalisation de ce projet m'a appris à travailler correctement en groupe et à communiquer avec les membres de mon groupe dans l'intention d'avoir un rendu final à la hauteur de ce qui nous a été demandé.

Néanmoins, nous avons fait face à plusieurs difficultés, tant sur le plan du temps que sur celui des exigences académiques, ce qui a retardé le début de notre projet.

Temps de travail : je considère avoir consacré plus de 60 heures à ce projet.

6.3 RIEU Valentin

Valentin Rieu s'est notamment occupé du client léger. Bien qu'on n'ai actuellement pas une version fonctionnelle de l'interface web. Il a réussi à implémenter une partie de la plateforme qui lui permet de s'authentifier directement sur la base de données en passant par les servlets avant de débuter une communication *websocket* en *javascript*. Par manque de temps et le projet n'étant pas abouti, nous n'avons malheureusement pas eu la possibilité de présenter les avantages qu'un tel principe aurait pu avoir.

L'interface web dont on dispose, bien que statique, montre ces capacités en programmation web, notamment en *html*, *css* et *javascript*. Des notions assimilées en cours de **Développement Web I** de la L2 Informatique ajoutées à des connaissances acquises en dehors du cadre scolaire y sont perceptibles.

6.4 ROMAIN Bruno

À cause de mon alternance et de mon obligation en entreprise, j'ai été moins présent que mes camarades sur le projet.

Dû à la charge de travail, nous n'avons pu commencer le projet que tard dans le planning. Nous avons eu aussi de multiples conflits dûs au fait que nous avons codé sur deux IDE différents.

Malgré ça, je trouve que nous avons tout de même bien travaillé sur le projet, sur le temps que nous nous sommes donné.

Ce que je trouve dommage sur ce projet est l'abstention d'utilisation de technologies qui auraient pu faciliter la communication entre les modules, et rendre l'application plus fiable.

Temps de travail : j'estime avoir passé environ 50 heures sur le projet.

7 Ressources utilisées

Voici les ressources qu'on a utilisées pour réaliser le projet :

- **IDE :** Eclipse, IntelliJ Idea
- **Outils de gestion :** Maven
- **Gestion de sources :** Git
- **Frameworks :** JQuery, nouislider, Ace

8 Conclusion

Notre plateforme n'étant pas aboutie, nous ne sommes pas satisfaits de cette dernière. Néanmoins, sur le temps qu'on y a consacré, nous sommes satisfaits de la qualité de ce que nous avons pu rendre. Nous y avons tous acquis des compétences et des connaissances essentielles qu'on pourra réutiliser plus tard.

Nous avons également beaucoup de pistes sur lesquelles travailler pour améliorer notre plateforme et la rendre fonctionnelle.

L'utilisation de technologies étudiées en cours nous a permis d'aborder un thème bien différent de ceux qu'on abordent dans les autres unités d'enseignement et nous donnera peut-être l'occasion de le réaborder mais avec des technologies plus avancées et donc un meilleur résultat.

9 Sitographie

- Developer Mozilla - *https://developer.mozilla.org/en-US/*
- JDK 20 Documentation - *https://docs.oracle.com/en/java/javase/20/*
- W3 Schools - *https://www.w3schools.com/*
- CodeJava - *https://www.codejava.net/*
- JAVASCRIPT.INFO - *https://javascript.info/*
- noUiSlider - *https://refreshless.com/nouislider/*
- Ace - *https://ace.c9.io/*