Rapport de stage TELECOM Nancy 2^e année

Editeur collaboratif pair-à-pair MUTE : Edition de texte riche

Camille Menou

Année 2016-2017

Stage de 2^e année réalisé dans l'équipe *COAST*

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Maître de stage : *Philippe Kalitine*
Encadrant universitaire : *Alexandre Parodi*

Déclaration sur l'honneur de non-plagiat

Je soussignée,
Menou Camille
Élève-ingénieure régulièrement inscrite en 2^e année à TELECOM Nancy
N° de carte d'étudiant(e) : 13538235
Année universitaire : 2016 - 2017
Auteur(e) du document, mémoire, rapport ou code informatique intitulé : Editeur collaboratif pair-à-pair MUTE : Edition de texte riche

Par la présente, je déclare m'être informée sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis consciente que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 24/08/2017

Signature :

Résumé :

MUTE est un éditeur de texte collaboratif temps-réel pair-à-pair. Basé sur le framework Angular, il s'agit d'une interface développée par l'équipe COAST, au sein du LORIA. En vue d'une utilisation prochaine au sein de l'INRIA, cet éditeur open-source a été rendu riche en ajoutant des fonctionnalités autour de l'ajout de style, de formules mathématiques et de listes. Ce travail est basé sur de l'interopérabilité de solutions open-source et sur des directives visant à améliorer l'interface et l'expérience utilisateur.

Abstract :

MUTE is a peer-to-peer collaborative real-time text editor. Based on Angular framework, it is developped by COAST team, in LORIA. For its usage inside INRIA, this open-source editor got rich by adding functionnalities around the addition of style, math formulas and lists. This work is based on interoperability of open-source solutions and on guidelines meant to improve the interface and the experience of the user.

Remerciements

Mes tout premiers remerciements sont adressés à M. Philippe Kalitine, ingénieur de recherche chez COAST et mon maître de stage. Merci pour la confiance, l'indépendance et l'expertise qui m'ont permises de mener à bien les tâches qui m'étaient confiées.

Merci à TELECOM Nancy pour la mise en place de cette période de stage afin que ses étudiants de seconde année puissent acquérir une expérience technique indispensable à la formation d'un ingénieur.

Merci à mon encadrant universitaire, M. Alexandre Parodi pour la supervision de ce stage et la lecture et notation de ce rapport.

De manière générale merci à toute l'équipe COAST pour sa convivialité. Venir travailler avec vous ce n'était plus du travail, mais l'exercice d'une passion commune dans une excellente ambiance !

Enfin, je tiens à remercier toutes les personnes qui se sont donnés la peine de relire ce rapport : M. Philippe Kalitine, M. Matthieu Nicolas et M. Gwendal Le Fur.

Table des matières

Introduction **1**

1 L'équipe COAST : présentation **2**
1.1 LORIA . . . 2
1.2 L'équipe COAST . . . 4

2 Problématique du stage : rendre un éditeur de texte riche **6**
2.1 Mise en contexte : MUTE . . . 6
2.2 Description et analyse du problème posé : apporter une édition riche . . . 7
2.3 Cahier des charges . . . 7

3 Réalisation et validation **9**
3.1 Auto-formation . . . 9
3.2 Environnement de travail . . . 9
3.3 Méthode de travail . . . 9
3.4 Cacher la syntaxe Markdown : intégration d'HyperMD . . . 11
3.4.1 Premières approches . . . 11
3.4.2 HyperMD . . . 12
3.5 Modifier le style . . . 13
3.5.1 Ajout de raccourcis clavier . . . 13
3.5.2 Création d'une barre d'outils . . . 14
3.6 Adaptation des appels à MathJax . . . 16
3.6.1 Analyse de la situation . . . 16
3.6.2 Un premier correctif . . . 16
3.6.3 Extraire les formules . . . 17
3.6.4 Doublons de marques : décalage des rendus . . . 17
3.6.5 Limites . . . 18
3.7 Création d'antisèches . . . 19
3.7.1 Markdown Cheat-sheet . . . 19
3.7.2 MathJax Cheat-sheet . . . 20
3.7.3 Inconvénients . . . 20
3.8 Tests . . . 22

4 Bilan du stage **23**
4.1 Avancement par rapport au cahier des charges prévu . . . 23
4.2 Retombées . . . 23
4.3 Développement futur . . . 23
4.3.1 Barre d'outil responsive . . . 23
4.3.2 Attendre les nouvelles versions de certaines dépendances ? . . . 23
4.3.3 Une nouvelle fonctionnalité : paramètres . . . 24
4.3.4 Améliorations des appels à MathJax . . . 24

Conclusion **25**

Webographie **26**

Glossaire **28**

Annexes **29**

Introduction

Dans le cadre de ma deuxième année dans l'école d'ingénieur TELECOM Nancy, j'ai effectué un stage technicien au Laboratoire lorrain de recherche en informatique et ses applications (LORIA) [1] à Vandœuvre-lès-Nancy. Durant dix semaines j'ai travaillé au sein de l'équipe COAST [2] dirigée par François Charoy, du 6 juin au 11 août de l'année 2017. Ce stage m'a été proposé par Philippe Kalitine, ingénieur en recherche en informatique chez COAST.

L'équipe travaille actuellement sur MUTE [3], un éditeur web de texte collaboratif pair-à-pair open source. Il s'agit d'une application semblable à Google Doc qui ne nécessite pas l'utilisation d'un serveur pour fonctionner. A terme, MUTE a pour vocation de devenir l'éditeur de texte principal du personnel Inria [4].

L'objectif de mon stage était de rendre l'éditeur riche, par exemple en intégrant des fonctionnalités autour de l'ajout de style dans les documents, d'images, de formules mathématiques, ou bien autour de l'exportation dans divers formats (PDF, HTML, etc.).

Dans le contexte de ce stage qui est celui de la recherche, le mode de travail ainsi que les enjeux sont différents du monde de l'entreprise. S'il n'y a pas un client à qui l'on doit livrer l'application, l'équipe COAST est néanmoins soumise à une deadline. En effet, l'équipe se prépare à faire une démonstration de MUTE lors de la quinzième European Conference on Computer-Supported Cooperative Work (ECSCW) [5] et qui aura lieu du 28 août au 1er septembre 2017 à Sheffield (UK). L'enjeu n'était donc pas des moindres et nécessitait un travail de qualité.

Ce document constitue le rapport de ce stage. Dans un premier temps, nous poserons le contexte en parlant de l'équipe COAST au sein du LORIA et de ses axes de recherche. S'ensuivront une mise en lumière de l'origine du projet ainsi que les objectifs du stage. Nous détaillerons ensuite l'ensemble des travaux réalisés, les résultats obtenus et les difficultés rencontrées. Enfin, nous dresserons un bilan de cette expérience technique en laboratoire de recherche et parlerons d'éventuelles améliorations à apporter au projet.

1 L'équipe COAST : présentation

1.1 LORIA

Le LORIA est une Unité Mixte de Recherche (UMR) [6] composée du Centre National de la Recherche Scientifique (CNRS) [7], de l'Université de Lorraine [8], et Inria. Créé en 1997, le LORIA est investi dans la recherche fondamentale et appliquée en sciences informatiques. Le LORIA fait aussi partie de la Fédération Charles Hermite [9] et du pôle scientifique Automatique, Mathématiques, Informatique et leurs interactions (AM2I) [10] de l'Université de Lorraine.

D'après son site Internet, en août 2017, ce ne sont pas moins de ving-neuf équipes, soit plus de quatre cents personnes qui travaillent au Loria. Les équipes sont réparties dans cinq départements :

— Département 1 : Algorithmique, calcul, image et géométrie
— Département 2 : Méthodes formelles
— Département 3 : Réseaux, systèmes et services
— Département 4 : Traitement automatique des langues et des connaissances
— Département 5 : Systèmes complexes, intelligence artificielle et robotique

On retrouve ces départements dans l'organigramme ci-après, extrait du site du LORIA.

Le conseil scientifique est composé du directeur, des deux directeurs adjoints et des responsables des cinq départements décrits plus haut. Tous les quinze jours, ils se réunissent afin d'assister le directeur dans les décisions concernant le LORIA.

Le conseil de laboratoire est composé de représentants de l'ensemble du personnel du LORIA. Tous les deux mois, il permet de débattre sur des questions de politique scientifique, organisation du laboratoire, budget, recrutement de personnels, formation, règlement intérieur, aménagement du temps de travail, conditions de travail et consultation des personnels...

Loria
Laboratoire lorrain de recherche en informatique et ses applications

ORGANIGRAMME 2016

DIRECTION / COMITE DE DIRECTION

Directeur : **Jean-Yves Marion**
Directeur adjoint : **Yannick Toussaint**
Responsable administratif : **Chantal Chrétien**
Assistante de direction : **Josiane Reffort**

Chargées de mission

-Cybersanté : M.-D. Devignes
-Sécurité : V. Cortier
-Systèmes cyberphysiques : F. Simonot-Lion

CONSEIL SCIENTIFIQUE

CONSEIL DE LABORATOIRE

AREQ - Assemblée des REsponsables d'éQuipes

COMITES

CHSCT *
CUMI *

(*) commun avec INRIA Nancy Grand-Est

SERVICE COMMUNICATION

Olivia Brenner
Marie Baron

SERVICE INFORMATIQUE de SOUTIEN à la RECHERCHE

Responsable : **Jean-Yves MARION**

Fabrice Sabatier
Patrice RINGOT
19 ingénieurs en CDD

SERVICE DE GESTION LORIA

Responsable :
Chantal CHRETIEN

Gestion du budget :
Julie Buquet
Jennifer Masoni

Suivi et gestion des départements :
Antoinette Courrier
Laurence Félicité
Delphine Hubert
Martine Kuhlmann
Christelle Levêque
Sylvie Musilli
Virginie Priester

Assistant de prévention :
Agnès Vidard

SERVICE INFORMATIQUE COMMUN

Responsable :
Jean-Michel ANTOINE

Patrick Aubier
Patrick Etienne
Antoine Falcone
Thierry Gartiser
Jean-Claude Giese
Lionel Maurice
David Mazé
Nathalie Pierre
Cédric Villa
Laurent Weinhard

Personnel INRIA en italique

29 EQUIPES/ 5 DEPARTEMENTS

Équipes de recherche communes avec INRIA Nancy Grand-Est

Département 1 : Algorithmique, calcul, image et géométrie

Responsable : Sylvain Lazard

- ABC - Y.Guermeur
- ADAGIO - I. Debled-Rennesson
- ALICE - B. Lévy
- CARAMBA - E. Thomé
- MAGRIT - M.O. Berger
- VEGAS - S. Lazard

Département 2 : Methodes formelles

Responsable : Horatiu Cirstea

- CARBONE - J.Y. Marion
- DEDALE - J.P. Jacquot/J. Souquiéres
- PAREO - H. Cirstea
- MOSEL - D. Méry
- TYPES - D. Galmiche
- CARTE - E. Jeandel
- PESTO - S. Kremer
- VERIDIS - S. Merz

Département 3 : Réseaux, systèmes et services

Responsable : Ye-Qiong Song

- OPTIMIST - A. Oulamara
- COAST - F. Charoy
- MADYNES - I. Chrisment

Département 4 : Traitement automatique des langues et des connaissances

Responsable : Bruno Guillaume

- CELLO - H. van Ditmarsch
- READ - A. Belaid
- SYNALP - C. Cerisara
- SMArt - K. Smaïli
- MULTISPEECH - D. Jouvet
- ORPAILLEUR - A. Napoli
- SEMAGRAMME - P. De Groote

Département 5 : Systèmes complexes, intelligence artificielle et robotique

Responsable : Patrick Henaff

- BISCUIT - B. Girau
- KIWI - A. Boyer
- CAPSID - D. Ritchie
- LARSEN - F. Charpillet
- NEUROSYS - L. Bougrain

1.2 L'équipe COAST

Comme on peut le constater sur l'organigramme du LORIA, l'équipe COAST, dirigée par M. François Charoy, se trouve dans le département associé aux Réseaux, systèmes et services. L'équipe travaille effectivement dans ce domaine mais également dans le calcul distribué. Ses thématiques de recherche sont les systèmes distribués et intergiciels [1].

Le projet de cette équipe est *"de définir des méthodes et des techniques permettant la construction d'applications collaboratives de confiance, basée sur une connaissance précise des algorithmes de réplication, sur la composition sure de service et sur une qualité de service qui peut être déduite et mesurée. Dans ce nouveau contexte, il est difficile de fournir des garanties prouvées sur les services. [Ils proposent] de baser [leur] travail sur une approche contractuelle et mesurable pour donner aux utilisateurs confiance dans les services qu'ils utilisent. Les axes de recherche de l'équipe COAST sont les suivants :*

- *La gestion sure et efficace de données collaborative à large échelle en terme de quantité de données et d'utilisateurs de ces données.*
- *La composition de services orientés données pour permettre la construction d'applications à l'échelle du Web et pour donner des garanties sur le fonctionnement de ces applications.*
- *Le support à la construction d'environnements collaboratifs pour lesquels on puisse déterminer un niveau de confiance et de sécurité"* [2].

L'équipe COAST est composée d'une chercheuse scientifique, de dix membres de facultés (professeurs ou assistants professeurs), de trois ingénieurs de recherche (dont mon maître de stage), de dix étudiants en thèse, de deux post-doctorants et de trois assistantes administratives. Pendant cette période de stage, nous étions également trois stagiaires de la promotion vingt-six de TELECOM Nancy. En Figure 1, un organigramme de l'équipe COAST.

1. Aussi appelé "middleware", un intergiciel est un logiciel tiers qui crée un réseau d'échange d'informations entre différentes applications informatiques
2. Sources : https://www.inria.fr/equipes/coast

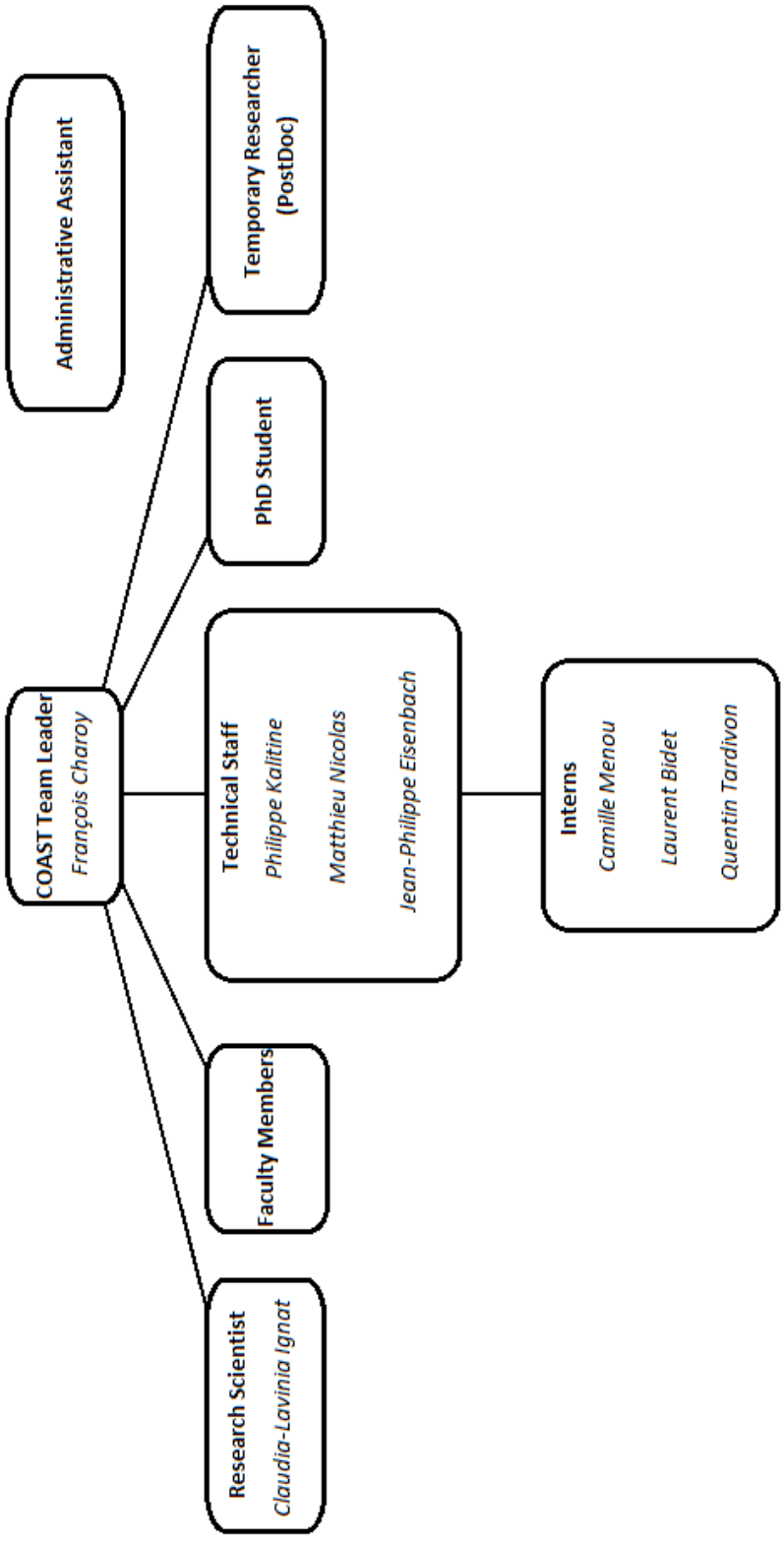


FIGURE 1 – Organigramme équipe COAST

2 Problématique du stage : rendre un éditeur de texte riche

2.1 Mise en contexte : MUTE

Origine du besoin A l'heure actuelle, au sein des entreprises, l'édition et le partage de fichiers entre collègues se fait très souvent sous forme de mail où le fichier est mis en pièce jointe. Cette pratique est très contraignante pour plusieurs raisons :

— c'est une procédure chronophage.
— les modifications apportées ne sont pas évidentes à voir.
— si beaucoup de personnes doivent modifier le même document et plus précisément un même paragraphe, la fusion de ces modifications devient très laborieuse.
— l'envoi d'un mail contenant des rapports confidentiels expose l'entreprise à des espionnages industriels.

Pour pallier ce problème, certains proposeraient tout simplement d'utiliser Google Doc, l'éditeur de texte de Google, qui a l'avantage d'apporter une édition collaborative d'un document et en temps-réel. Malheureusement, Google Doc utilise une architecture dite, centralisée : un serveur stocke le document et écoute les modifications des collaborateurs. De cela découlent trois inconvénients majeurs empêchant la mise en place de cette solution en entreprise :

— l'utilisation de l'éditeur ne passe pas à l'échelle.
— le fait que le document soit stocké sur les serveurs de Google ne plaît pas aux entreprises pour une question de sécurité (vol de données) mais aussi de sûreté (la sauvegarde du document peut être compromise si les serveurs de Google subissent une panne).
— l'édition d'un document devient impossible si les serveurs de Google sont en difficulté ou si notre connexion internet devient instable.

Afin de développer une plate-forme collaborative permettant de répondre à ces besoins, le projet OpenPaaS : :NG [11] a été créé. Financé par BpiFrance [12], ce projet open-source français est réalisé en collaboration avec les sociétés Linagora [13], XWiki SAS [14], Nexedi [15] et le laboratoire d'informatique de l'école polytechnique.

A cette intention, l'expertise de l'équipe COAST a été requise. Leur rôle a été de montrer qu'il est possible de mettre en place
— un réseau pair-à-pair capable de mieux passer à l'échelle que Google Doc.
— une collaboration sûre (sans pertes de données).

COAST a donc créé l'API indépendante mute-core [16], constituée de deux sous-projets :

Mute-structs Mute-structs [17] est un projet Node.js qui propose une implémentation de l'algorithme CRDT (Conflict-free replicated data type) LogootSplit [18]. Cet algorithme est issu d'une thèse provenant de l'équipe COAST et sert à la fusion des contributions des collaborateurs sans générer de doublons ou de perte de données.

Netflux Nextflux [19] est une API permettant l'établissement d'un réseau pair-à-pair.

MUTE Les ingénieurs de l'équipe développent l'application MUTE qui met donc en valeur ces résultats de recherche. Multi User Text Editor (MUTE) est donc une interface web basée sur l'API CodeMirror et qui se sert de l'API mute-core. MUTE est donc un éditeur web collaboratif temps-réel pair-à-pair.

Si MUTE n'avait au début vocation qu'à ne servir que de plateforme de démonstration pour Netflux et mute-structs, MUTE doit maintenant devenir utilisable au sein de l'INRIA.

2.2 Description et analyse du problème posé : apporter une édition riche

Afin de rendre l'éditeur utilisable au sein d'INRIA, il est nécessaire de le rendre riche. Il s'agissait donc d'ajouter des nouvelles fonctionnalités autour de l'édition pour améliorer l'expérience utilisateur.

Il fallait pouvoir ajouter du style au texte (gras, italique...), des gestions de listes, des liens hypertextes, des images ou des formules mathématiques. Il fallait proposer plusieurs manières d'ajouter ce style : écriture de la syntaxe nécessaire à la main, à l'aide de raccourcis clavier, ou grâce à des éléments visuels cliquables.

Ces manières d'ajouter doivent être simples d'utilisation, intuitives et rester dans la norme de ce qui existe en termes d'édition, de traitement de texte, afin de ne pas perturber l'expérience utilisateur.

De plus, étant donné que LogootSplit ne traite que du texte brut, il est nécessaire que les solutions apportées ne nécessitent pas un transfert d'informations aux collaborateurs par un moyen autre que du texte.

2.3 Cahier des charges

Le cahier des charges du projet a été défini comme étant les tâches GitHub (plus communément appelées par le terme anglais "issues") auxquelles j'étais assignée. Les détails nécessaires m'ont été fournis à l'oral au moment opportun. De part le caractère open-source du projet, le cahier des charges a été rédigé en anglais. Avant de passer à la lecture du cahier des charges, il est nécessaire d'expliquer ce qu'est Markdown. Markdown est un langage de balisage qui permet d'ajouter du style au texte (gras, italique, listes, liens, titres...). Un langage de balisage est un langage de description de données utilisant des balises. Soit la syntaxe **exemple**. Les astérisques sont les balises, "exemple" la donnée à décrire. En Markdown, cette syntaxe signifie que la donnée "exemple" doit être mise en gras. Ci-dessous, la liste des issues que je devais réaliser pendant mon stage :

As a User I want to :

see a partially styled (inline rendered style) Markdown text in the editor : les balises correspondant à de la syntaxe Markdown doivent être camouflées après interprétation, une fois que l'édition du texte stylisé est terminée ou qu'on clique sur un autre élément que celui-ci. De même, les balises doivent réapparaître lorsque l'utilisateur met le focus sur un élément contenant du style. Plus précisément, quand on écrit **exemple**, tant que le curseur se trouve dans ce texte, on doit voir les balises. Mais quand le curseur n'est plus sur cette zone, on ne doit voir que "exemple", en gras.

see a rendered version of my document : MUTE aurait dû être divisée en deux parties : à gauche l'utilisateur écrirait le texte souhaité, à droite il aurait accès à un rendu où la syntaxe aurait été interprétée. Cette issue est devenue obsolète du fait de la précédente (voir explications dans la partie Réalisation et validation). Si l'issue précédente n'était pas réalisable immédiatement, il aurait fallu faire cette issue, étape intermédiaire entre ce qui existait au début du stage, et ce qui existe maintenant.

have an outline view for Markdown : l'utilisateur doit pouvoir accéder à une table des matières du document à côté de ce dernier. Lors de la création d'un titre (balises #, ##, etc...), le texte associé doit apparaître dans la table des matières. Lors du parcours du document, le titre correspondant à la partie actuellement visible du document doit être mis en valeur dans cette table (mis en gras par exemple).

add basic styles (bold, italic...) to the document using keyboard shortcuts : l'utilisateur doit pouvoir ajouter/enlever du style en utilisant des raccourcis claviers intuitifs, par exemple Ctrl+B pour le gras.

add basic styles (bold, italic...) to the document using toolbar : au moment d'une sélection de texte, une barre d'outils doit apparaître pour que l'utilisateur puisse ajouter/enlever le style à la souris.

see my math formula rendered : l'utilisateur doit pouvoir écrire une formule mathématique et qu'elle ait le même rendu que s'il l'avait écrite à la main.

include images into my documents : tout est dans l'intitulé.

to add some "End of Page" character : au bout d'un certain nombre de lignes, la page doit se terminer et une nouvelle doit être créée.

consult Markdown cheat-sheet : l'utilisateur doit pouvoir rapidement accéder à la syntaxe de Markdown

consult MathJax cheat-sheet : l'utilisateur doit pouvoir rapidement accéder à la syntaxte de MathJax

be able to export the document to another document format (PDF, plain text) : tout est dans l'intitulé.

Les issues à propos des cheat-sheets et de l'exportation comportaient le label [3] "optionnal" pour dénoter une priorité basse. Les autres issues ont été classées par ordre de priorité décroissante, comme ci-dessus.

L'ensemble de mon travail était régi par un besoin fonctionnel incontournable : si des fonctionnalités visent à modifier le contenu du document alors le contenu doit effectivement être modifié, et ce pour n'importe quel collaborateur du document. Un autre besoin, non négligeable, était que les développements marchent à la fois sur Chrome et sur Firefox.

3. Aussi appelé 'tag', il s'agit d'un mot permettant de décrire ou de mettre une issue dans une catégorie. Par exemple le label 'bug' signifie que l'issue décrit le dysfonctionnement d'une fonctionnalité.

3 Réalisation et validation

3.1 Auto-formation

Le stage a d'abord commencé par une période d'auto-formation à base de Massive Open Online Courses (MOOCs), de vidéos de conférences et d'articles écrits par des développeurs/ingénieurs de la communauté informatique. A cette occasion, et pendant les onze premiers jours je me suis donc formée en JavaScript et en TypeScript (JavaScript typé).

Le projet utilisant le framework AngularJS j'ai également lu à ce sujet grâce à la Tour of Heroes [20], projet Angular écrit pour un tutoriel disponible sur le site officiel d'Angular [21].

3.2 Environnement de travail

La partie édition de MUTE est basée sur une API qui s'appelle CodeMirror [22]. Il s'agit d'un éditeur de texte écrit en JavaScript et pensé pour les navigateurs. Ce projet assez populaire (12,364 stars et 3000 forks au 4 août 2017) est agrémenté d'addons permettant d'étendre les fonctionnalités de CodeMirror au besoin.

L'ensemble du projet est hébergé sur un répertoire GitHub, un gestionnaire de version.

Il m'a été conseillé d'utiliser l'IDE (Integrated Development Environment) VisualStudio Code pour développer. Particulièrement adapté au langage JavaScript, VisualStudio permet de voir très facilement les fichiers modifiés par rapport au dernier commit et de faire des merges très simplement.

Le projet utilise le framework Angular. Ce framework est centré autour de la notion de composant. Un composant gère une partie de la vue (l'interface). Un composant est constitué :
- d'un fichier JavaScript (en l'occurrence TypeScript) qui contient les données à afficher ainsi que les fonctions gérant leur comportement,
- d'un fichier HTML qui permet de structurer les données de l'interface,
- d'un fichier CSS qui contient les éléments de style à appliquer aux données.

Le projet utilise npm [23] afin de gérer ses dépendances.

3.3 Méthode de travail

Deux outils m'ont quotidiennement aidée à m'organiser et à prendre des notes pour faciliter l'écriture de ce rapport : l'outil de gestion de projet GitHub, ainsi qu'un journal quotidien que j'ai tenu selon le principe du bullet journal [24].

GitHub workboard Il s'agit d'un tableau référençant l'ensemble des issues m'étant attribués, de la documentation et ma roadmap. Une roadmap est le terme employé pour parler de l'organisation des issues dans le temps. L'organisation se fait de manière hebdomadaire. Vous trouverez un aperçu de ce tableau en Annexe A.

C'était un excellent moyen pour accéder rapidement à mes issues et cela permettait également à mon maître de stage de pouvoir jeter un oeil à mon avancement.

Journal quotidien Afin de noter un maximum de choses pour écrire ce rapport mais aussi pour noter ma réflexion, ne pas oublier de faire quelque chose ou juste mieux m'organiser, j'ai pris l'initiative de tenir un journal quotidien de mon travail. A cette occasion j'ai repris quelques principes des bullet journals comme par exemple le fait de noter systématiquement les tâches à faire en organisant des checklists. Pour lier le workboard au journal, j'ai noté chaque jour sur quelle(s) issue(s) j'ai travaillé avec un code couleur pour retrouver plus rapidement des notes que j'aurais prises à l'intention de l'une d'elles.

Compléter une issue Avant de commencer une issue, j'en informais mon maître de stage qui me donnait certaines pistes de réflexion et de recherche à explorer avant de commencer.

Ainsi, en toute autonomie, je menais des recherches les plus exhaustives possibles pour trouver des solutions open source ou juste des méthodes que j'aurais pu reproduire dans notre projet. Nous décidions ensuite ensemble du meilleur choix.

Après le développement de la solution, je présentais le résultat, relevant les limites et recevant des détails à rajouter au développement. Pour certaines issues, notamment pour les raccourcis clavier et la barre d'outils, je référençais sur l'issue correspondante l'ensemble des fonctionnalités présentes. En Annexe B, un exemple.

L'avantage de cette pratique est qu'il sera possible pour les personnes qui maintiendront mon code de voir s'il y a eu des régressions ou tout simplement de voir ce qui existe déjà et ce qu'il reste à faire.

Communication avec l'équipe Lors des premières semaines, chaque vendredi, l'équipe se réunissait pour faire état de l'avancement de chacun. L'équipe utilise également Slack [25] pour communiquer.

Abordons maintenant de manière plus concrète les différentes étapes de réalisation du cahier des charges.

3.4 Cacher la syntaxe Markdown : intégration d'HyperMD

Afin d'apporter du style à l'éditeur (gras, italique, etc.), l'équipe a décidé d'utiliser le langage à balise Markdown [26]. A mon arrivée, MUTE était déjà configuré pour interpréter ce langage et l'on obtenait le rendu de la Figure 2.

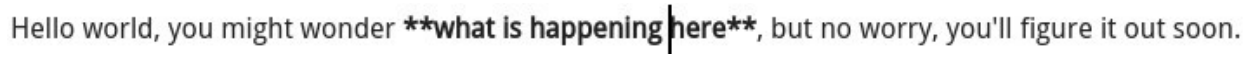


FIGURE 2 – Syntaxe Markdown apparente

On constate que le style est appliqué mais que les balises sont encore visibles. Comme indiqué dans le cahier des charges, l'une de mes tâches a été de faire en sorte de cacher la syntaxe.

3.4.1 Premières approches

Une approche à la main Il s'agissait de la première issue que j'avais à faire. Dans un premier temps, j'ai essayé de résoudre le problème avec "les moyens du bord".

Dans mon analyse de la situation, j'ai remarqué que l'utilisation de la syntaxe Markdown au sein de l'éditeur modifiait le DOM [27]. Le DOM est une interface de programmation qui, entre autres, organise le contenu d'une page HTML en une structure d'arbre qu'il est possible de parcourir et modifier. Si on reprend l'exemple de l'image ci-dessus, on obtient ce DOM, en Figure 3

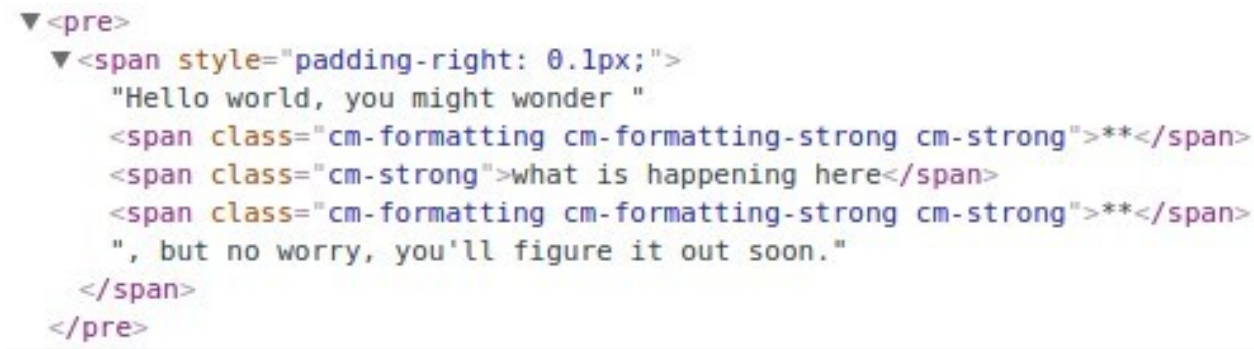


FIGURE 3 – DOM modifié

De ce fait, je pensais faire une recherche des noeuds du DOM dont les classes contiendraient des éléments en rapport avec le style CSS apporté par Markdown. Une fois ce style récupéré il aurait fallu l'appliquer uniquement au "what is happening here". Pour faire disparaître les balises sans pour autant les perdre, j'aurai essayé d'appliquer l'attribut CSS visibility en le mettant à "hidden" aux deux spans contenant les balises.

Cependant, cette solution paraissait trop lourde car elle nécessitait un parcours constant du document. De plus, si une personne avait été en train d'écrire, l'édition en aurait été gênée.

Mon maître de stage m'a alors suggéré de rechercher des API développées en ce sens à intégrer dans notre projet. Mais aucune n'a montré de résultats concluants.

3.4.2 HyperMD

Après plusieurs essais avec différentes API, mon maître de stage m'a dirigée vers HyperMD [28], projet datant de Janvier 2017, et dont la démo [29] paraissait remplir les objectifs de cette première issue et plus.

Après avoir installé HyperMD dans notre projet, nous avons constaté qu'en plus de cacher la syntaxe, les liens étaient stylisés, les images étaient prises en compte, ainsi que les formules mathématiques, grâce à MathJax [30].

MathJax est une librairie open-source, qui, à partir d'une formule respectant une syntaxe particulière, permet de générer un rendu mathématique. HyperMD utilise cette librairie de la manière suivante : il faut encadrer la formule de "$" ou de "$$". Une fois que l'on a fini d'écrire la formule, celle-ci est interprétée. Les dollars disparaissent, et le texte est remplacé par un élément MathJax appelé "jax" qui affiche le rendu stylisé de la formule. Si on clique sur la formule, le jax disparaît, on repasse à une édition de la formule avec les dollars. Lors des premiers tests d'utilisation d'HyperMD, un dysfonctionnement de MathJax m'est apparu : si un collaborateur A modifie une formule, le collaborateur B ne voit pas de modifications de son côté. Cependant s'il clique sur la formule, la formule qui s'affiche en mode édition (avec les dollars) se trouve être la formule écrite par A. Quand il quitte le mode d'édition, MathJax refait le rendu, et B a le même rendu que chez A. Comme il est primordial que dans un contexte collaboratif chaque utilisateur ait la même version du document, j'ai pris l'initiative de créer une issue GitHub décrivant le problème, en y ajoutant le label 'bug'.

J'ai aussi pris l'initiative de créer une issue pour créer une cheat-sheet pour MathJax, à l'instar de Markdown, afin d'améliorer l'expérience utilisateur.

3.5 Modifier le style

Afin d'ajouter du style à un texte, il existe un autre moyen que de simplement écrire la syntaxe Markdown à la main. Dans cette partie, nous verrons deux de ces moyens : les raccourcis clavier et les barres d'outils.

3.5.1 Ajout de raccourcis clavier

Un raccourci clavier est une combinaison de touches permettant d'effectuer une action. A chaque raccourci est associé une action unique. Un raccourci très connu : Ctrl+Z, permet d'annuler la dernière modification effectuée. Dans un éditeur de texte, certains raccourcis ne fonctionnent que si du texte est sélectionné.

Les éléments de style pour lesquels il semblait le plus pertinent de créer un raccourci clavier étaient : le gras, l'italique, le fait de rayer un texte et la création d'un lien. Il a ensuite fallu chercher quel est le meilleur raccourci pour ces éléments. Par meilleur, il faut entendre : utilisé dans un maximum d'éditeurs, facile à réaliser et facile à retenir.

J'ai jugé qu'il était plus adéquat qu'un raccourci modifie le texte sélectionné plutôt que le DOM. Ainsi, HyperMD n'aurait plus qu'à interpréter les changements effectués.

J'ai ensuite créé deux fonctions. La première gère le gras, l'italique et le rayage, la deuxième, les liens.

Gras, italique, rayé Pour ces raccourcis j'ai donc associé une liste d'expressions régulières. On utilise une liste car il existe plusieurs syntaxes pour le même style. Par exemple, on peut écrire __exemple__ ou **exemple** pour appliquer du gras.

On parcourt la liste d'expressions régulières pour voir si le texte sélectionné contient déjà le style qu'on essaye de lui appliquer.

- Si le style n'est pas détecté, on lui applique le style associé au raccourci clavier en ajoutant la syntaxe de part et d'autre de la sélection.
- Si le style est détecté, on enlève la syntaxe associée au raccourci clavier. Pour ce faire, on parcourt la sélection des extrémités vers le centre pour enlever la syntaxe correspondante. Juste enlever les extrémités n'aurait pas suffit car il est possible que d'autres styles (et donc d'autres éléments de syntaxes) soient appliqués au texte.

Liens Un lien en syntaxe Markdown s'écrit de la manière suivante : [texte du lien](url). J'ai pris l'initiative que la partie URL de cette syntaxe soit sélectionnée. Ainsi, l'utilisateur n'a plus qu'à compléter la syntaxe avec l'URL de son choix. J'ai aussi pris en compte le fait que l'utilisateur puisse faire une sélection ne correspondant qu'à une partie de la syntaxe seulement. Le principe algorithmique est le suivant :

- Si la sélection "matche" très exactement l'expression régulière, alors on sélectionne la partie URL.
- Sinon, on parcourt la sélection.
  - Si on a détecté le style CSS associé aux liens (information récupérée grâce au DOM), on continue de parcourir jusqu'à repérer l'expression "(url)" pour pouvoir ensuite la sélectionner.
  - Sinon le lien n'existe pas, il faut le créer.

Limites Pour chacun de ces deux cas, une limite est apparue.

— L'utilisateur peut faire une sélection grâce à un double clic. Il se trouve que si l'utilisateur double clique __exemple__, l'entièreté du texte sera sélectionnée, alors que pour **exemple**, seul "exemple" sera sélectionné. Ainsi, si on est dans le cas **exemple** et qu'on utilise le raccourci clavier associé au gras, on ne va pas enlever le gras mais en rajouter. On va alors avoir ****exemple**** avec "**exemple**" en gras. Pour réduire la probabilité que ce cas arrive, les raccourcis claviers associés au gras et à l'italique ont été paramétrés pour utiliser les tirets plutôt que les astérisques.

— Il existe un cas pour lequel l'utilisation du raccourci pour les liens n'a pas le comportement attendu. Si on reprend [texte du lien](url), il s'avère que la partie "rl)" ne comporte pas le style CSS associé aux liens. Ainsi, si l'utilisateur sélectionne uniquement cette partie du lien, il va être considéré que la sélection n'est pas un lien, donc un nouveau lien va être créé, au lieu que l'URL soit sélectionnée. On va donc avoir [texte du lien](u[texte du lien](url)rl).

3.5.2 Création d'une barre d'outils

Une nouvelle fonctionnalité de MUTE est l'ajout d'une barre d'outils. Elle sert à ajouter du style (gras, italique, rayé) mais aussi à créer les liens, les citations, les listes et à sélectionner la taille du texte en vue de créer un titre grâce à des boutons ou des menus déroulants. Il a été souhaité qu'elle apparaisse juste au dessus d'une sélection quand une sélection de texte est faite, et disparaisse quand la sélection disparaît. Pour plus d'esthétique, j'ai fait en sorte que le milieu de la barre d'outils corresponde au milieu de la sélection. Cette barre d'outil a été désactivée pour les supports mobile car elle n'est pas bien adaptée à ce genre de support. Il est prévu qu'une barre d'outils similaire soit développée pour apparaître au bas de l'écran, de manière fixe, sur support mobile.

Le composant barre d'outils N'ayant trouvé aucune API open-source correspondant aux besoins, il a fallu créer un composant Angular pour intégrer cette barre d'outils. Pour ce faire, j'ai pris connaissance de l'existance de Material Angular [31]. Il s'agit de composants Angular représentant des éléments d'interface comme des boutons, des menus déroulants, etc. La barre d'outils est composée de cinq "toggle buttons" et de deux menus déroulants. Le "toggle button" est un bouton à deux états : actif (quand on a cliqué dessus), inactif (après un deuxième clic).

Pour qu'un élément d'interface ne soit pas dérangeant, soit intuitif et donne envie d'être utilisé, il m'a été conseillé de lire les directives Angular sur les éléments d'interface REFERENCE. Elles énoncent des principes à respecter et dénoncent les faux pas à ne pas commettre, que ce soit pour la charte des couleurs, la disposition d'un élément ou la manière dont ce dernier doit se déplacer dans l'interface. Ainsi, la barre d'outil est dans les tons orangés, couleur secondaire de la charte des couleurs de MUTE et qui est censée indiquer à l'utilisateur que ces éléments d'interface sont à l'origine d'actions. De même, l'apparition et la disparition de la barre d'outils ont été configurées.

En Figure 4, un aperçu de cette barre d'outils.

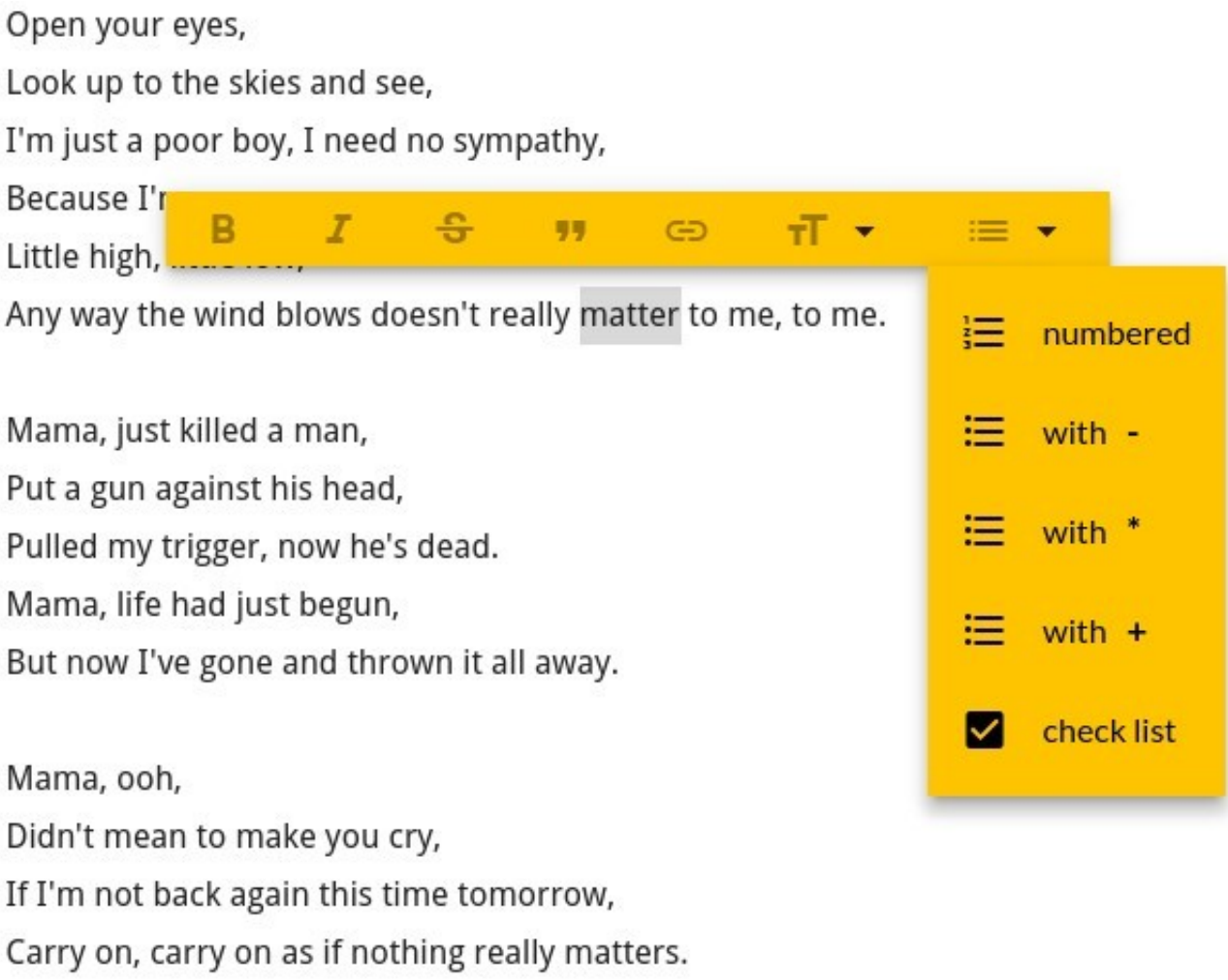


FIGURE 4 – Barre d'outils

Limites Une première limite réside en la manière dont sont développés les "toggle buttons" de Material Angular. Il existe un scénario où l'activation de ces boutons se désynchronise d'avec la situation actuelle. Mettons que l'on décide de sélectionner un mot et de le mettre en gras grâce au raccourci clavier. J'ai fait en sorte d'analyser et de récupérer quels styles sont présents dans une sélection, afin qu'à l'apparition de la barre d'outils, les "toggles buttons" soient activés ou non pour le(s) style(s) qui correspond(ent). Une fois le gras appliqué avec le raccourci clavier, le bouton correspondant au gras est dans l'état activé. Si cependant on souhaite retirer le gras grâce au bouton, le gras est bel et bien retiré mais le bouton reste néanmoins dans l'état activé.

Une deuxième limite réside en la manière dont CodeMirror gère les copier/coller. Pour CodeMirror, une ligne est représentée par une balise "pre" dans le DOM, caractésirée par une classe "CodeMirrorline". A chaque création de ligne par la touche "Entrée" une nouvelle ligne est créée et dans l'interface et dans le DOM. Mais, si la ligne est créée car la ligne précédente est remplie, CodeMirror ne considère pas ça comme une nouvelle ligne. Ainsi, si on décide de coller un texte de plusieurs lignes dans l'éditeur, il nous apparaît comme un paragraphe de plusieurs lignes. Mais, dans le DOM, on trouve que le texte entier est stocké dans une seule balise "pre". Cela implique que, peu importe où la sélection sera faite dans le paragraphe, la barre d'outils apparaîtra uniquement au dessus de la première ligne du paragraphe.

3.6 Adaptation des appels à MathJax

MathJax est une librairie offrant une grande diversité de symboles mathématiques qui correspond parfaitement au besoin décrit dans le cahier des charges. Cependant, j'ai relevé un "bug" dans le fonctionnement de cette librairie. Résoudre ce bug ne faisait pas partie de mes issues, mais j'ai pris l'initiative d'essayer de le résoudre. Cependant, la multitude de scénarios possible dans un contexte temps-réel collaboratif fait que j'ai dû me pencher sur le problème qui va suivre à plusieurs reprises car je n'ai pas pu penser à toutes les configurations d'utilisation de ces formules au sein d'un document. Il est d'ailleurs certains que des cas limites ne soient pas encore traités.

3.6.1 Analyse de la situation

Lors de l'intégration d'HyperMD au projet, j'ai remarqué que si un collaborateur A décidait de modifier une formule, un collaborateur B ne verrait le changement que s'il mettait le focus sur cette formule. Même si cela ne reste qu'un bug d'affichage ne causant pas de divergences [4], j'ai proposé à mon maître de stage d'essayer de le corriger avant de m'occuper des cheat-sheets.

Dans un premier temps, j'ai essayé de comprendre comment il était possible que l'éditeur du collaborateur B arrive néanmoins à mettre à jour la formule correctement. Il aurait été plus logique qu'au moment de focus ladite formule rien ne se passe de son côté. J'en ai conclu que la syntaxe de la formule mise à jour était donc bien transmise aux collaborateurs.

J'ai pensé que l'information pouvait être stockée dans le DOM. Mais je n'ai constaté aucun changement dans le DOM.

Pour mieux comprendre comment HyperMD fait appel à MathJax j'ai étudié le code source. Quand une expression du type "$ ... $" ou "$$ ... $$" est détectée, HyperMD créé une marque. Au sens de CodeMirror, une marque est ce qui est utilisé pour "marquer" une partie d'un texte en lui attribuant une classe CSS spécifique. Au moment de créer cette marque, il est possible de spécifier des options. Entre autres, HyperMD décide que l'élément DOM correspondant à cette zone de texte doit être remplacé par un élément span contenant un élément script. Or, grâce à l'élément script, MathJax est en mesure de générer un rendu. Ce rendu est nommé dans la documentation MathJax "jax".

En consultant la documentation de CodeMirror, j'ai constaté qu'il était possible d'avoir accès à l'ensemble des marques. C'est dans cette liste que j'ai retrouvé la syntaxe de la nouvelle formule changée par A (cf en Annexe). En mettant le focus sur la formule, le jax est retiré et la marqué est cassée : on retourne à un aspect d'édition de type "$ ... $". Or, comme la marque a été modifiée par A et transmise à B, le contenu de "$ ... $" affiché chez B sera bien ce qui se trouve chez A. Une fois que le focus est enlevé, la marque est recréée, MathJax est appelé, et le jax est donc mis à jour. J'en conclus donc qu'il faut forcer MathJax à reprocéder au rendu des formules.

3.6.2 Un premier correctif

Dans un premier temps j'ai procédé comme suit : on récupère les marques et on récupère les jax. On parcourt ces deux listes en parallèle. On récupère le texte contenue dans la marque. A

4. On parle de divergence quand, pour un même document, les collaborateurs de ce document n'en ont pas la même version à un instant t.

l'aide d'une expression régulière on récupère la formule. On demande à MathJax de s'exécuter avec la nouvelle formule sur la jax correspondante.

HyperMD utilisant le même système de marques pour les liens et les images, il ne faut travailler que sur les marques dédiées aux maths. La distinction s'est facilement faite car chacune avait un nom de classe différent.
Pour simuler le fait que les rendus soient sensibles à la modification en temps-réel, j'ai utilisé une fonction JavaScript permettant d'exécuter du code selon un intervalle de temps. Je l'ai programmé à cent millisecondes dans un premier temps. Cela ne causait aucun ralentissement dans l'utilisation de l'éditeur.

3.6.3 Extraire les formules

Au hasard de l'utilisation de l'éditeur pendant que je travaillais sur une autre issue, j'ai constaté une anomalie lorsque, sur une ligne, une formule est accompagné d'autres éléments (texte ou autre) :

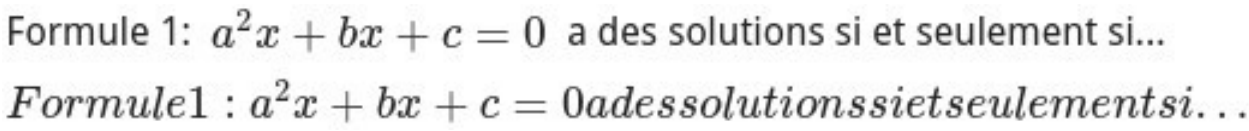


FIGURE 5 – Anomalie de rendu

Il semble que MathJax essaye de faire un rendu de tout le texte contenu dans la ligne au lieu de simplement la formule. En regardant de plus près le contenu de la marque il s'avère qu'elle prenait effectivement toute le texte de la ligne en considération.

Pour pallier ce problème j'ai donc créé une fonction pour récupérer dans une marque les éléments du texte répondant aux expressions "$ ... $" ou "$$ ... $$". Ainsi, on passe à MathJax la formule au lieu du texte contenu dans la marque.

Pour le cas où l'utilisateur soit en train d'ajouter une formule à une ligne en contenant déjà une (imaginer que le texte est "$ formule 1 $ $"), cet algorithme considère que les deux derniers dollars constituent une formule. Cela entraîne des problèmes de rendus qui sont très perturbants pour l'utilisateur en train rédiger. Pour pallier ce problème j'ai donc empêché le rafraîchissement d'une ligne contenant un nombre impair de $.

3.6.4 Doublons de marques : décalage des rendus

Encore pendant que je travaillais sur une autre issue, j'ai remarqué un décalage dans les rendus MathJax. Par exemple : si sur la ligne 1 se trouvent les formules A et B, sur la ligne 2 la formule C et que l'utilisateur focus la formule B, alors sur la ligne 2 on verra s'afficher le rendu de la formule B. Ce décalage n'est pas transmis aux autres collaborateurs. L'illustration de cette situation est dépeinte dans les Figures 6 et 7

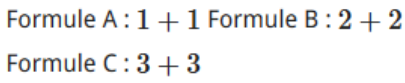


FIGURE 6 – Situation initiale

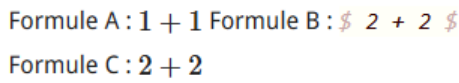


FIGURE 7 – Décalage du rendu

Il s'avère que si sur une ligne il y a deux formules, chacune de ces formules va générer une marque. On aura donc deux marques aux contenus identiques. Ainsi, quand la formule B est 'cassée', la marque de la formule A subsiste. Comme on extrait les formules depuis les marques, et comme on itère sur les marques et les jaxs en parallèle, on applique donc à la formule C le rendu de la formule B.

Pour pallier ceci j'ai repensé mon code pour pouvoir isoler la ligne contenant une formule en train d'être éditée pour pouvoir toujours garder synchronisés la liste des marques et la liste des jaxs. Toujours en itérant sur les marques, on compte le nombre de formules qu'elle contient :

- S'il est égal à un, nous sommes dans un cas disons classique, et nous allons juste faire le rendu. Pour être sûr de mettre à jour la bonne jax par rapport à la liste des marques, on met en place un compteur symbolisant index de la prochaine jax à mettre à jour.
- S'il y a plusieurs formules dans cette marque, on stocke ce nombre et on passe à la marque suivante :

  - Si la marque est différente et qu'on n'a pas passé autant de marques que le nombre de formules détectées, c'est que l'une d'elle est en édition. On ne met rien à jour et on augmente le compteur de la prochaine jax à mettre à jour.
  - Si la marque est identique mais qu'on n'a pas passé autant de marques que de formules détectées alors on ne fait rien et on passe à la marque suivante.
  - Si la marque est identique et qu'on a passé le bon nombre de marques alors aucune des formules n'est en train d'être éditée et on peut alors faire un rendu.

3.6.5 Limites

Cette problématique à elle seule aurait dû constituer une issue plus importante que simplement une issue avec le label 'bug'. Cependant je devais aussi avancer sur d'autres fonctionnalités donc mon travail n'est pas aussi complet que ce que je l'aurai souhaité. Par exemple, la solution mise en place actuellement bloque le rendu des jaxs des lignes contenant un nombre impair de '$'. Or il est possible que ce cas subvienne. Néanmoins, la mise à jour des formules est toujours possible si on focus la formule.

L'utilisation de MathJax a montré quelques difficultés lors d'un test réalisé avec neuf navigateurs. Des fois les formules disparaissaient complètement. Il est difficile de reproduire les scénarios où MathJax a dysfonctionné.

Avec un intervalle de cent millisecondes, il arrivait des fois que toutes les formules s'affichent en deux fois si on cassait/reformait les marques trop vite. Ce bug ne touche pas aux marques et disparaissait très rapidement. Pour régler le problème, l'intervalle est de maintenant trois secondes. Si cela sacrifie un peu le caractère 'temps-réel' cela offrira peut-être une meilleure stabilité au fonctionnement de MathJax en collaboration.

3.7 Création d'antisèches

Afin d'améliorer l'expérience des utilisateurs de MUTE, deux issues ont été créées pour ajouter des cheat-sheets à l'éditeur, littéralement des antisèches. Ces antisèches contiendraient des rappels de syntaxe pour Markdown et MathJax. Nous avons décidé avec mon maître de stage que ces éléments seraient incrustés à droite du document, au même niveau que les informations le concernant (localisation du stockage et collaborateurs), afin que l'utilisateur puisse les avoir à portée tout en gardant accès à l'éditeur. Il m'a été suggéré de réfléchir à une solution à base d'onglets à intégrer au niveau du composant nommé très justement "right-side".

Via mes recherches pour la barre d'outils, je me suis souvenue qu'Angular Material proposait un module Tabs (onglets). J'ai donc immédiatement travaillé sur cette solution. J'avais alors deux manières d'organiser les informations : soit avoir trois onglets (Details, Markdown, MathJax), soit avoir deux onglets (Details, Cheat-sheets) avec dans Cheat-sheets deux onglets (Markdown et MathJax). J'ai opté pour la deuxième solution, car trois onglets prenaient trop de place, le nom des onglets était tronqué. De plus cela fait plus de sens de stocker toutes les cheat-sheets dans un même onglet.

3.7.1 Markdown Cheat-sheet

Pour le contenu, et après avoir regardé des équivalents sur d'autres éditeurs, j'ai opté pour une disposition sur deux colonnes avec à gauche le rendu, à droite la syntaxe associée. L'idée que j'avais était d'avoir un rendu intuitif et raccord avec le reste de l'interface et les guidelines Angular.

J'ai tout d'abord essayé de travailler avec les tables HTML, mais le rendu était peu satisfaisant. Le fait de voir un tableau à cet endroit paraissait trop scolaire, le rendu était trop chargé. J'ai englobé le contenu dans un card, un autre composant Angular. Pour comprendre à quoi cela correspond, la zone d'édition du document, ressemblant beaucoup à feuille blanche, est un card. Cela m'a donné l'idée qu'il serait intéressant pour l'utilisateur de voir ce que le rendu donne réellement sur le document. Ainsi, et pour donner de la structure à l'antisèche, j'ai mis le card en gris clair, puis y ait inséré six autres cards, eux en blanc. Chacun de ses cards représentera les catégories suivantes : les headers, les styles, les listes, les images, les liens et le code informatique. J'ai créé ces catégories en m'inspirant de cette cheat-sheet Markdown [32]. Précédant chacun de ces cards, un titre en gris foncé, taille h2 permet de nommer chacune de ces catégories et sert de séparateur naturel entre chacun des cards.

Afin que les rendus soient fidèles à ce que génère HyperMD, j'ai eu l'idée d'importer la feuille CSS liée à HyperMD dans la feuille CSS du right-side. Ainsi, pour chaque texte auquel je voudrais appliquer le rendu HyperMD je n'aurai qu'à appeler la classe correspondante. Mais le style ne s'appliquait pas. Après plusieurs essais, j'ai décidé recréer des classes dans la feuille de style de right-side pour reproduire le rendu HyperMD. Cette solution a marché mais cela a produit une duplication de code, il y a certainement moyen de faire autrement.

Après présentation à mon maître de stage, celui ci m'a suggéré que le fond gris clair soit de même couleur que l'arrière plan de l'application. Ainsi, les cards ressortent vraiment, sont à la même élévation que la zone d'édition et donnent un effet "post-it" qui est, je pense, vraiment parlant pour l'idée de mémo. L'utilisateur a donc une réelle idée du rendu que peut avoir ces différentes syntaxes.

Enfin, mon maître de stage a également suggéré que la cheat-sheet ait son propre composant Angular, sous-composant de right-side.

3.7.2 MathJax Cheat-sheet

Pour cette antisèche, je suis partie sur le même concept d'apparence et de structuration.

Dans un premier temps, mon maître de stage et moi-même désirions que les rendus des formules soient générés par MathJax. MathJax étant utilisé uniquement par HyperMD, je n'avais jamais eu besoin de l'importer au sein de notre projet. Cette tâche seule s'est étalée sur trois jours. Pour ne pas pas perdre trop de temps, en parallèle, je commençais de mettre en place le composant qui allait contenir l'antisèche. Finalement, après que mon maître de stage soit venu m'aider à mettre en place MathJax, nous avons constaté qu'une particularité de MathJax était bloquante : en effet, MathJax charge les formules globalement, sur l'ensemble de l'interface. Plus précisément : la liste de 'jax' mentionnée en 3.6 est unique à l'application. Ainsi, les rendus se retrouvaient décalés : la première formule dans l'éditeur avait son rendu dans l'antisèche, et toutes les autres formules se trouvaient décalée d'un cran.

Nous avons donc conclu qu'il serait plus simple que les rendus dans l'antisèche soient des images provenant d'impressions d'écrans. L'avantage à cela est que le rendu correspondra tout à fait à ce que MathJax produit dans l'éditeur, de plus, cela fait moins de formules à charger lors de l'affichage de l'antisèche.

3.7.3 Inconvénients

HyperMD prévoit que tout texte collé dans l'éditeur à partir du presse-papier soit automatiquement échappé. Par exemple, *Hello world !* devient *Hello world !*. Il en va de même si l'utilisateur choisit de copier/coller la syntaxe d'un élément l'intéressant de l'antisèche vers l'éditeur. En soit ce n'est pas vraiment une mauvaise chose, car l'utilisateur pourra effectuer des manipulations avant d'enlever les caractères d'échappement et que le rendu ne se fasse. De plus les sections copiées ne sont pas très longues. Mais cela nécessite quand même une manipulation supplémentaire. Sans compter que pour des utilisateurs moins au fait de l'utilisation de l'éditeur cela peut poser un problème de compréhension.

Un autre inconvénient vient de l'utilisation des Tabs. Dans l'onglet Details se trouve l'affichage des contributeurs du projet. Il se trouve que le contenu de cet onglet refusait de prendre tout l'espace. Ainsi, le contenu était réduit à un très petit espace, qui, par conséquent, générait une barre de défilement, ce qui n'était pas très esthétique. Mon maître de stage a pris en charge la résolution de ce problème. Cela a eu un effet sur les antisèches, qui ne sont plus contenues dans un card gris, mais sont juste disposées sur le fond de l'application. Le résultat reste très sensiblement le même.

Le dernier inconvénient vient de la manière dont sont implémentées les Tabs d'Angular Materiel. La première fois que l'on décide de consulter les cheat-sheets, les deux sous-onglets se découvrent et l'on voit par défaut la cheat-sheet Markdown. Or, l'onglet Markdown n'est pas marqué comme étant sélectionné. Le résultat de ce travail est visible Figure 8.

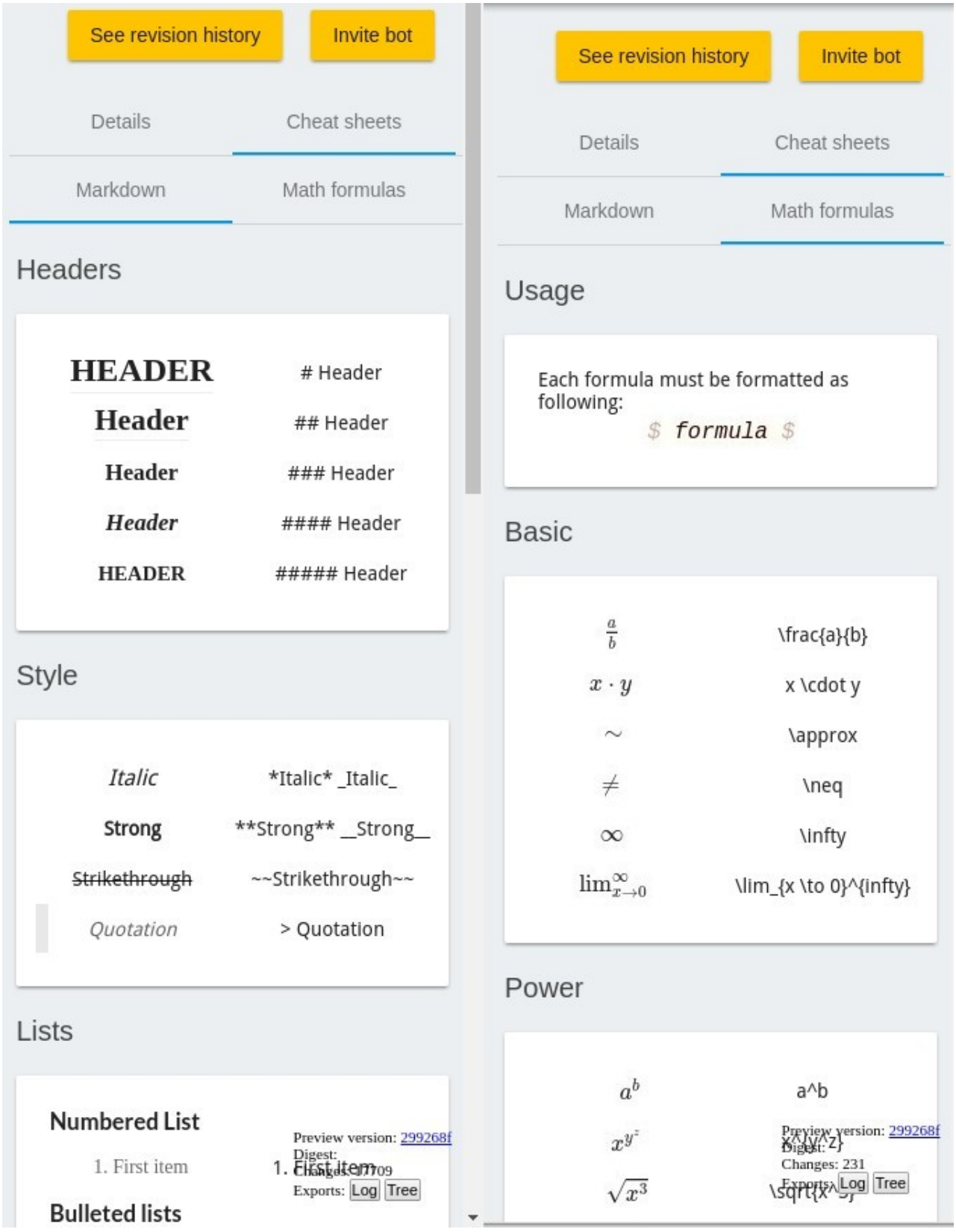


FIGURE 8 – A gauche l'antisèche Markdown, à droite l'antisèche MathJax

3.8 Tests

Au cours du stage, il est arrivé que mon aide soit requise pour des tests. En effet, si l'on peut plus ou moins tester la robustesse de notre développement en ouvrant plusieurs navigateurs à la fois, il est impossible de reproduire l'édition simultanée d'un document par plusieurs collaborateurs. C'est pourquoi mon maître de stage, d'autres collègues, voire même moi-même demandions un coup de main à d'autres. Ces tests permettaient aussi que d'autres mettent en avant des scénarios où le développement de l'un ou de l'autre était incomplet.

De plus, chaque compte rendu des réunions hedbomadaires était pris via MUTE. Nous pouvions ainsi tester la dernière démo, ce qui permettait de mettre en avant des oublis, des bugs ou des précisions sur les cas d'utilisations.

4 Bilan du stage

4.1 Avancement par rapport au cahier des charges prévu

Si on se réfère au cahier des charges, la majorité des tâches a été accomplie. Il faut comprendre que l'on m'a attribués les issues en rapport avec la thématique de mon stage. Il y avait toujours possibilité d'ajouter d'autres tâches en rapport avec mon travail. En somme, il n'y a pas de réelle fin à ce travail. Mon travail a été apprécié dans son ensemble.

4.2 Retombées

MUTE et donc mon travail sont disponibles sur le serveur de production [5]. L'éditeur sera utilisé chaque semaine lors des réunions hebdomadaires de l'équipe. De plus, l'éditeur sera présenté à l'ECSCW à Sheffield.

L'algorithme qui effectue la fusion des collaborations ne travaille qu'avec du texte brut. Ainsi, mon travail a permis de mettre en avant la possibilité de transférer du style et même des formules mathématiques grâce à ce seul algorithme en travaillant avec des API de parsing de texte. Si cela n'avait pas marché, il aurait fallu développer une autre structure pour transférer le style.

4.3 Développement futur

4.3.1 Barre d'outil responsive

Comme précisé dans la partie correspondante, la barre d'outil n'est pas responsive. Même s'il s'agit d'un détail, il reste toujours très apprécié et est un gage de qualité au sein d'une application.

4.3.2 Attendre les nouvelles versions de certaines dépendances ?

Certaines limites de développement évoquées dans ce rapport tiennent dans la manière dont sont développées certaines librairies.

HyperMD Le projet HyperMD et MUTE présentaient une issue en commun : la création d'une barre d'outils. Au cours de mon analyse des fonctionnalités proposées par HyperMD j'ai dressé une liste des issues ouvertes sur ce projet et qui peut impacter le nôtre :
— interprétation des paragraphes
— interprétation des tableaux
Après être entré en communication avec le propriétaire de ce projet il s'avère qu'HyperMD n'est pas sa priorité numéro une. Il conviendrait donc de traiter ces cas nous-mêmes.

MathJax Une particularité de cette librairie qui la rend unique face aux librairies utilisées dans MUTE est qu'elle est globale à l'application. Il serait préférable qu'on puisse la configurer pour la rendre globale à un document particulier. Depuis 2015, il est possible de voir sur le répertoire GitHub de ce projet qu'une version 3 est en cours de réflexion. Il semble donc qu'il se déroule une certaine période avant d'obtenir quelque chose de plus stable. Il serait peut-être envisageable de trouver une autre librairie qui remplisse les besoins escomptés.

5. https://www.coedit.re

Angular Material Au contraire de MathJax, le projet Angular Material semble plus actif. Il est donc très probables que les deux limites énoncées à l'encontre de l'utilisation d'Angular Material soient corrigées prochainement, mais elles représentent très certainement une priorité faible.

4.3.3 Une nouvelle fonctionnalité : paramètres

A l'issue de ce stage, beaucoup de nouvelles fonctionnalités ont été mises en place. Toujours soucieuse de l'expérience utilisateur, j'ai imaginé qu'il serait intéressant de mettre en place un système de paramétrage de certaines options.

Ainsi on pourrait laisser à l'utilisateur le soin de pouvoir activer/désactiver la barre d'outils, et les previews MathJax. Mais aussi, quand beaucoup de formules sont chargées au sein d'un même document, si la configuration de l'ordinateur de l'utilisateur met en cause le bon fonctionnement de MUTE, demander à augmenter l'intervalle de rafraîchissement des formules. Dans le même soucis de performances, s'il y a beaucoup de collaborateurs sur le même document, ou tout simplement par confort, animer le curseur d'un collaborateur seulement quand il écrit, voire complètement les désactiver.

Pour mettre en place cette fonctionnalité, on pourrait mettre en place un bouton paramètres dans le menu gauche de MUTE. Ce bouton pourrait faire apparaître une boîte de dialogue comportant la liste des différents paramètres cités précédemment. En face de ces éléments, on pourra mettre en place un objet d'Angular Material. Pour les activations/désactivations, des slige toogles [33]. Pour régler la fréquence des rafraîchissements et la complexité des animations, des sliders [34].

4.3.4 Améliorations des appels à MathJax

En attendant une version plus stable de MathJax, et si aucune autre librairie viendrait à se substituer à celle-ci, nous avons discuté avec mon maître de stage de certaines dispositions qui pourraient être mises en place pour limiter son dysfonctionnement.

Dans un premier temps, il serait possible de ne plus demander le rendu des formules selon un intervalle de temps mais seulement quand une modification est perçue. S'il m'était possible de pouvoir détecter une modification, je n'ai cependant pas eu l'idée de comment récupérer cette information d'HyperMD et de la transmettre à MUTE.

Grâce à l'API Netflux, il est possible d'ajouter au réseau pair-à-pair un "bot". L'idée première de cette fonctionnalité est de permettre à un utilisateur de pouvoir stocker son document sur la base de données OVH (où est hébergé MUTE) en plus de la base de données native du navigateur internet. Ce bot est relié à tous les autres pairs et peut aussi servir à faire des calculs qu'on lui aurait délégué. Ainsi, il serait possible de déléguer le rendu des formules MathJax à ce bot qui n'aurait plus qu'à transmettre le résultat aux autre pairs. Ainsi, les navigateurs pourraient s'affranchir de toutes ces requêtes MathJax.

Il en va sans dire que s'il était possible d'associer ces deux dispositions, on obtiendrait une utilisation moins prenante en ressources et certainement plus stable.

Conclusion

L'objectif de ce stage a été d'apporter des fonctionnalités autour de l'ajout de style au sein de MUTE. Grâce à des solutions open-source ou à des directives reproductibles au sein du projet, nous avons vu comment l'éditeur a évolué au cours de ces dix semaines. Nous avons également vu les limites de ces solutions.

Après discussion avec mon maître de stage, celui-ci m'a fait prendre conscience que les entreprises développent elles-mêmes les outils qui leur permettent de remplir leurs besoins. Donc, au contraire des entreprises, on réalise que les équipes de recherches, même si elles sont en première ligne dans les avancées technologiques et théoriques, sont aussi les premières à pâtir des limites de ces dernières. En l'occurrence ici, on constate que le développement des solutions que j'ai apportées sont soumises au développement des API, donc au travail de beaucoup de personnes. Certaines d'entre elles développent ces API dans un but de loisir et non professionnel. De plus, mon maître de stage m'a indiqué que le principe d'une équipe de recherche est de montrer qu'il est possible de mettre en place des solutions novatrices. En l'occurrence nous avons montré qu'il est possible d'ajouter la gestion de style, de formules mathématiques et de listes tout en gardant l'algorithme de fusion initial LogootSplit.

Le fait de travailler en équipe de recherche m'a fait réaliser deux choses. La première est qu'il est difficile d'atteindre le résultat parfait dans le contexte de la recherche et des projets open-source. J'attends mon expérience en entreprise en troisième année pour confirmer ou infirmer cette impression. La deuxième chose est qu'évoluer dans ce contexte, ou en tout cas dans l'équipe COAST, permet de travailler en toute autonomie. Chaque idée ou piste de recherche de solution est valorisée, ce que je trouve très motivant.

Webographie

[1] *LORIA 2016*. URL : http://www.loria.fr/fr/.

[2] *Présentation - COAST - Inria*. URL : https://www.inria.fr/equipes/coast.

[3] COAST TEAM. *mute*. URL : https://github.com/coast-team/mute.

[4] *Inria - Inventeurs du monde numérique*. URL : https://www.inria.fr.

[5] *The 15th European Conference on Compute-Supported Cooperative Work, Sheffield (UK) August 28 - September 01 2017*. URL : https://ecscw2017.org.uk/.

[6] *Unité Mixte de Recherche - Wikipédia*. URL : https://fr.wikipedia.org/wiki/Unit%5C%C3%5C%A9_mixte_de_recherche.

[7] *Centre national de la recherche scientifique*. URL : www.cnrs.fr.

[8] *Université de Lorraine*. URL : www.univ-lorraine.fr.

[9] *Fédération Charles Hermite*. URL : http://www.fr-hermite.univ-lorraine.fr/.

[10] *Angular*. URL : https://www.univ-lorraine.fr/sites/www.univ-lorraine.fr/files/node_files/publics/users/francoi48/2015/01/plaquette_am2i_vdef_5nov.pdf.

[11] *OpenPaas - An open source Entreprise Social Platform*. URL : http://open-paas.org/.

[12] *Bpifrance - Servir l'Avenir*. URL : http://www.bpifrance.fr/.

[13] *Linagora.com*. URL : https://linagora.com/.

[14] *Xwiki SAS*. URL : http://www.xwiki.com/fr/.

[15] *Nexedi SAS*. URL : http://www.nexedi.com/fr.

[16] COAST TEAM. *mute-core*. URL : https://github.com/coast-team/mute-core.

[17] COAST TEAM. *mute-structs*. URL : https://github.com/coast-team/mute-structs.

[18] Luc ANDRE. « Préservation des Intentions et Maintien de la Cohérence des Données Répliquées en Temps Réel ». In : *Annalen der Physik* (2016). DOI : http://docnum.univ-lorraine.fr/public/DDOC_T_2016_0089_ANDRE.pdf.

[19] COAST TEAM. *mute*. URL : https://github.com/coast-team/netflux.

[20] *Angular - Tutorial: Tour of Heroes*. URL : https://angular.io/tutorial.

[21] *Angular*. URL : https://angular.io.

[22] *CodeMirror*. URL : https://codemirror.net/.

[23] *npm*. URL : https://www.npmjs.com/.

[24] *Bujoter en français - Jamais sans mon carner*. URL : http://bulletjournal.fr/.

[25] *Slack: Where work happens*. URL : https://slack.com/.

[26] *Markdown - Wikipédia*. URL : https://fr.wikipedia.org/wiki/Markdown.

[27] *Document Object Model - Wikipédia*. URL : https://fr.wikipedia.org/wiki/Document_Object_Model.

[28] LAOBUBU. *laobubu/HyperMD: WYSIWYG Markdown Editor for browsers. Breaks the Wall between writing and preview*. URL : https://github.com/laobubu/HyperMD.

[29] LAOBUBU. *HyperMD Playground*. URL : https://demo.laobubu.net/.

[30] *MathJax*. URL : https://www.mathjax.org/.

[31] *Angular Material - Tabs*. URL : https://material.angular.io/components/tabs/overview.

[32] *Markdown Cheatsheet - adam-p/markdown-here Wiki*. URL : https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet.

[33] *Angular Material - Slide toggle*. URL : https://material.angular.io/components/slide-toggle/overview.

[34] *Angular Material - Slider*. URL : https://material.angular.io/components/slider/overview.

Glossaire

Annexes

Annexe A

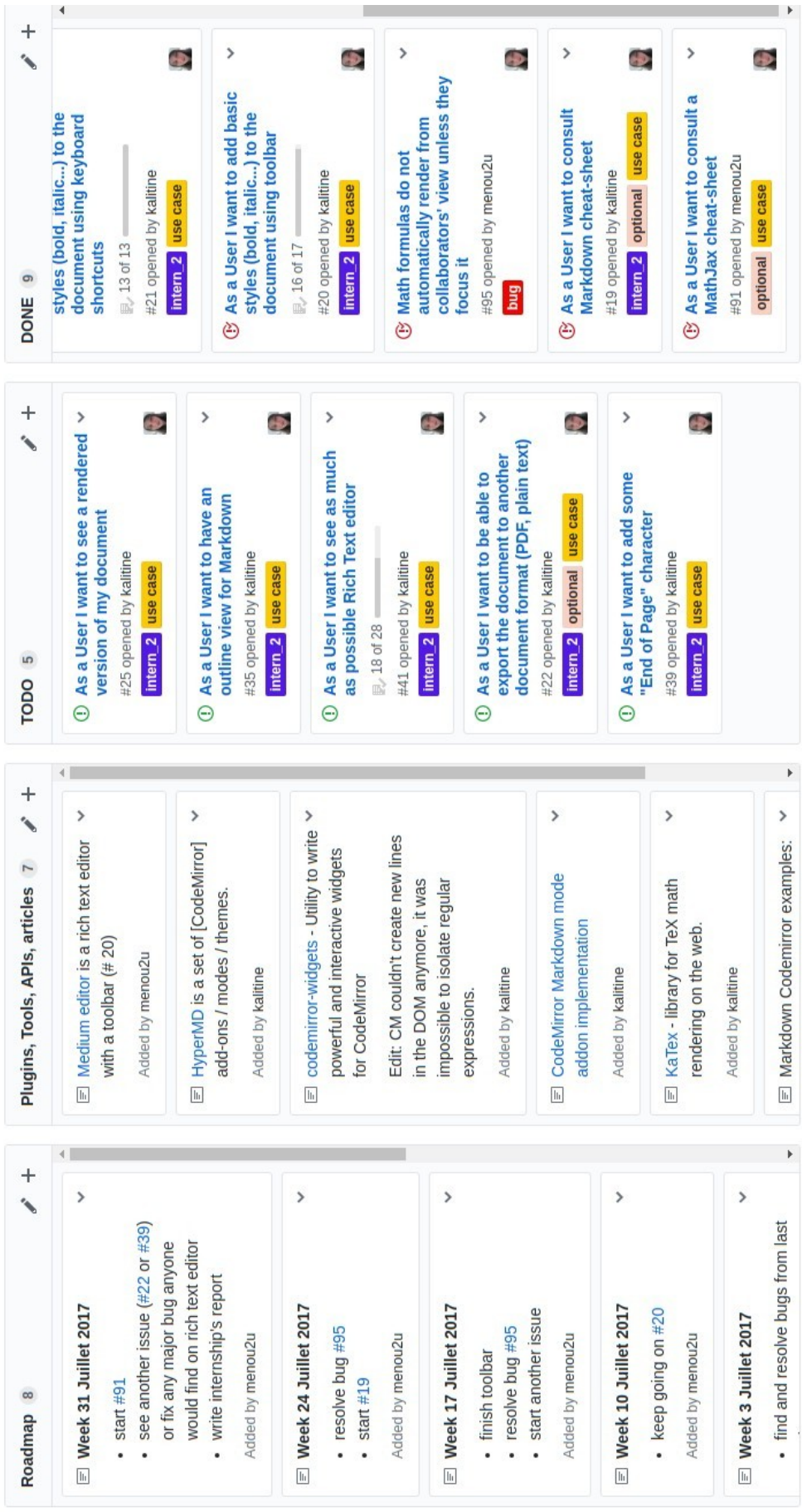
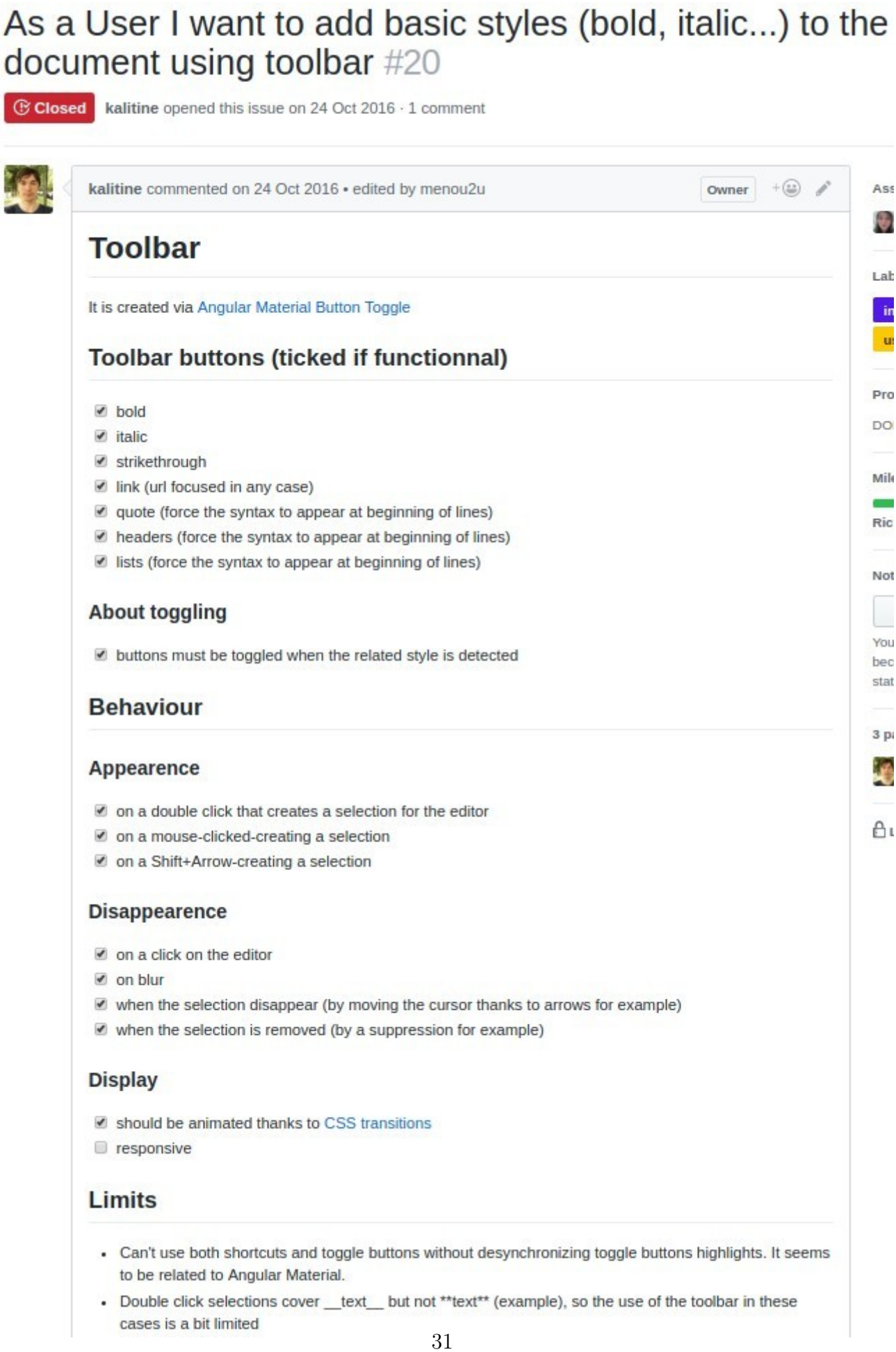


FIGURE 9 – Workboard du projet Rich Text Editor

Annexe B



As a User I want to add basic styles (bold, italic...) to the document using toolbar #20

Closed kalitine opened this issue on 24 Oct 2016 · 1 comment

kalitine commented on 24 Oct 2016 • edited by menou2u Owner

Toolbar

It is created via Angular Material Button Toggle

Toolbar buttons (ticked if functionnal)

- [x] bold
- [x] italic
- [x] strikethrough
- [x] link (url focused in any case)
- [x] quote (force the syntax to appear at beginning of lines)
- [x] headers (force the syntax to appear at beginning of lines)
- [x] lists (force the syntax to appear at beginning of lines)

About toggling

- [x] buttons must be toggled when the related style is detected

Behaviour

Appearence

- [x] on a double click that creates a selection for the editor
- [x] on a mouse-clicked-creating a selection
- [x] on a Shift+Arrow-creating a selection

Disappearence

- [x] on a click on the editor
- [x] on blur
- [x] when the selection disappear (by moving the cursor thanks to arrows for example)
- [x] when the selection is removed (by a suppression for example)

Display

- [x] should be animated thanks to CSS transitions
- [ ] responsive

Limits

- Can't use both shortcuts and toggle buttons without desynchronizing toggle buttons highlights. It seems to be related to Angular Material.
- Double click selections cover __text__ but not **text** (example), so the use of the toolbar in these cases is a bit limited

FIGURE 10 – Issue associée à la barre d'outils

Mots-clefs :

interopérabilité, collaborative-editing, Angular