

Vérificateur orthographique et grammatical de PDF

Rapport final du projet

Sebastien BRO – Pauline KELDER – Marc LAURENT – Marie MAINGUY
– Anaïs MANGOLD – Nicolas REY

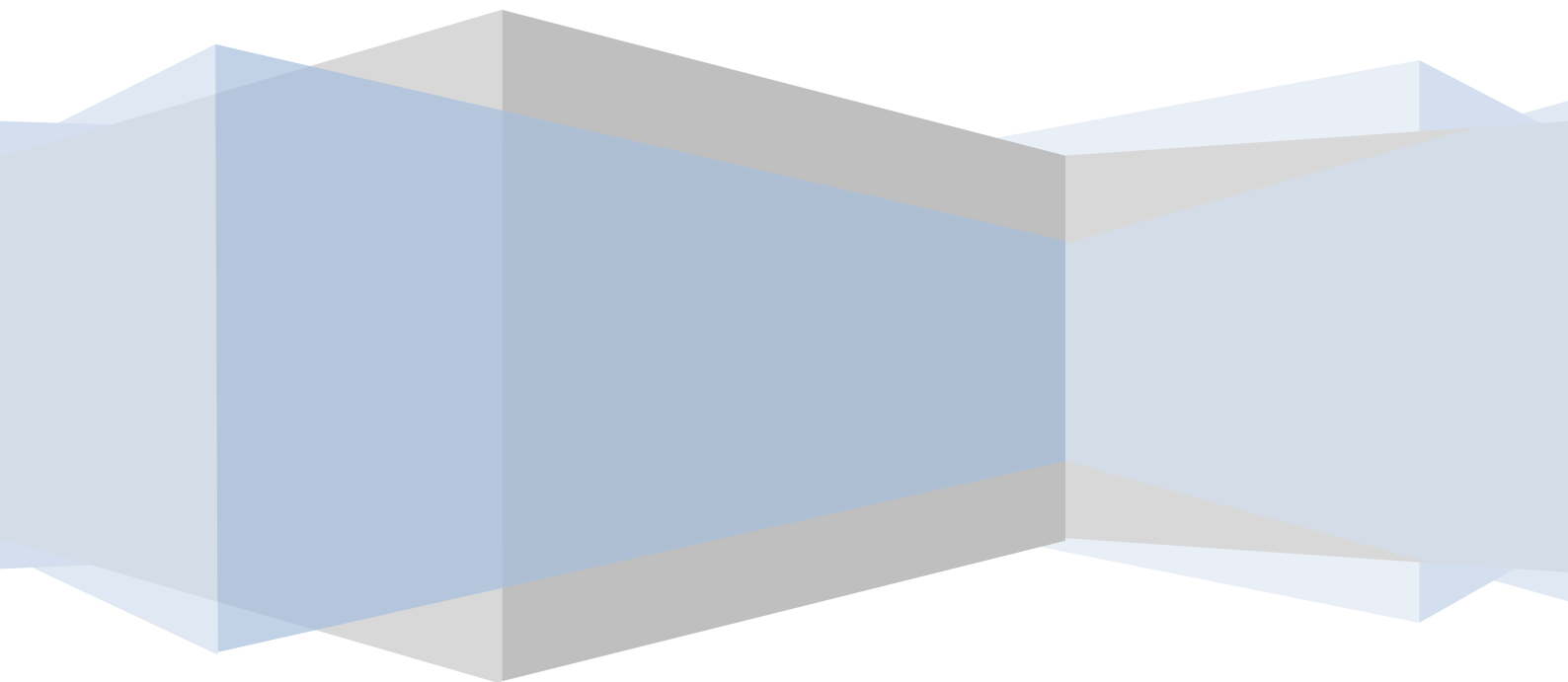


Table des matières

I. Présentation du projet – Rappel du cahier des charges	4
Présentation du projet	4
Engagement de l'équipe projet – Cahier des charges.....	4
II. Organisation de l'équipe projet	5
Planning de travail suivi.....	5
Répartition du travail (choix de travail par binôme)	8
Management.....	9
Podio.....	10
Dépôt git.....	11
III. Déroulement du projet.....	12
Etat de l'art.....	12
Extracteur de pdf.....	12
Correcteur orthographique	13
Correcteur grammatical	13
Difficultés rencontrées	14
Architecture et articulation des outils.....	14
Amélioration de l'extraction	14
Suppression des en-têtes et pieds de pages	14
Gestion du multi-colonnes	15
Réalisation de l'interface web	16
Principe.....	16
Fonctionnement	16
Relier l'outil à l'interface	17
Améliorations diverses apportées à l'outil.....	18
Extraction du texte	18
Correction orthographique:	18
IV. Résultat final : présentation de l'outil.....	19
Ecart avec le cahier des charges.....	19
Diagrammes descriptifs de l'outil final.....	19
Diagramme de cas d'utilisation	19
Diagramme de séquences	21
Diagramme composants.....	22
Diagramme de déploiement	22

Modularité du code.....	22
Licence.....	23
V. Pistes d'améliorations envisagées	24
Sélectionner les zones à corriger	24
Principe.....	24
Pistes technologiques.....	24
Des comptes utilisateurs avec un historique	24
Principe.....	24
Pistes technologiques.....	26
Traitement des formules mathématiques	26
Principe.....	26
Pistes technologiques.....	26
Amélioration de la correction en ligne après traitement.....	27
Principe.....	27
Pistes technologiques.....	27
Accélération du traitement de la correction.....	27
Principe.....	27
Principe.....	28
VI. Conclusion.....	29
VII. Webographie.....	30
VIII. Table des figures	30

I. Présentation du projet – Rappel du cahier des charges

Présentation du projet

L'objectif de notre projet était de réaliser un vérificateur orthographique et grammatical de fichiers au format PDF. Il s'agissait donc de réaliser une interface web qui permettrait à l'utilisateur de charger un fichier de type PDF depuis son ordinateur, puis de récupérer en sortie un fichier comportant la liste des erreurs orthographiques et grammaticales détectées au sein du document.

Dans un premier temps, notre équipe projet devait réaliser un état de l'art des différents outils d'extraction de texte, de vérificateurs orthographiques, ainsi que de vérificateurs grammaticaux déjà existants, afin d'avoir une vue globale des outils pouvant être utilisés au sein de notre vérificateur. Et dans un second temps, nous devions relier ces différents outils afin d'obtenir un correcteur complet et mettre en relation ce nouvel outil avec notre interface web.

Engagement de l'équipe projet – Cahier des charges

Nous nous sommes engagés dans un premier temps à réaliser une première version du correcteur permettant la vérification de fichiers PDF dits "simples", c'est à dire mono-colonne, ne contenant ni images, ni en-tête, ni numéro de page (en bas de page). Le changement de paragraphe est considéré comme étant possible pour un fichier "simple".

En fonction de l'avancement du projet, notre équipe s'est engagée à inclure un maximum de fonctionnalités à cette première version:

- Traitement de documents à plusieurs colonnes
- Traitement de documents contenant des en têtes, etc.

Enfin, l'équipe s'attachera dans un premier temps à renvoyer à l'utilisateur la liste des mots du fichier contenant une erreur, en précisant sa position dans le texte. Il n'est pour l'instant pas prévu de proposer une liste de corrections possibles. Ceci fera l'objet d'une fonctionnalité à rajouter en fin de projet, en fonction de l'avancée de celui-ci. Nous avons plusieurs possibilités de sortie du correcteur, la première étant celle que nous nous sommes engagés à faire par défaut :

- Page HTML avec ligne de l'erreur et le mot ou groupe de mots concernés
- Fichier TXT avec mots soulignés
- Fichier PDF avec mots soulignés

En fonction de la rapidité de développement de l'interface Web nous incluons les fonctionnalités suivantes, la première étant celle que nous nous engageons à faire par défaut :

- Une page web avec un champ pour soumettre le PDF.
- Une page web avec des comptes utilisateurs et un historique par utilisateur
- Une page web avec des comptes utilisateurs et un historique organisé par version des PDF soumis si un même PDF passe par le correcteur plusieurs fois par utilisateur

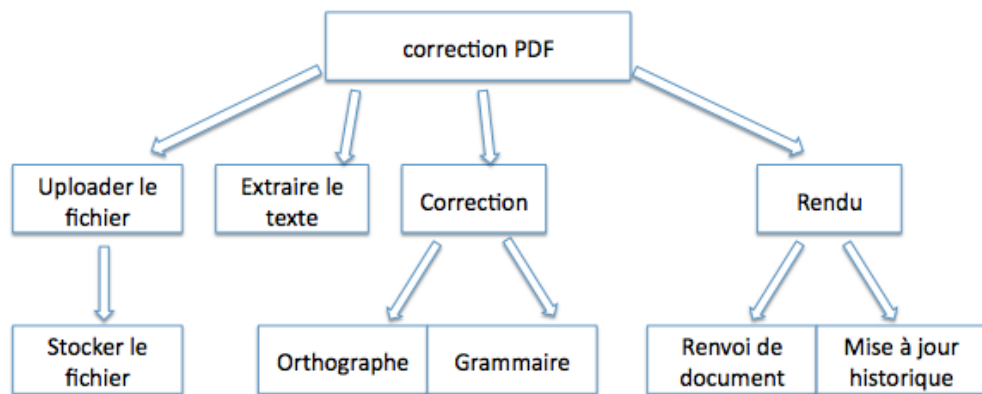


Figure 1 - Diagramme de spécification

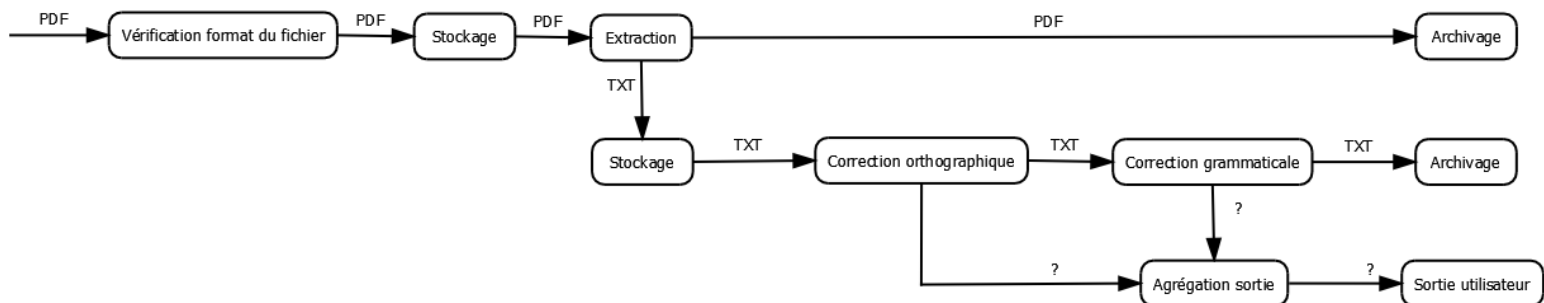


Figure 2 - Cycle de vie

Java étant le langage utilisé pour le serveur, il a donc été utilisé majoritairement durant le projet.

II. Organisation de l'équipe projet

Planning de travail suivi

Le premier planning établi lors de l'écriture du cahier des charges.

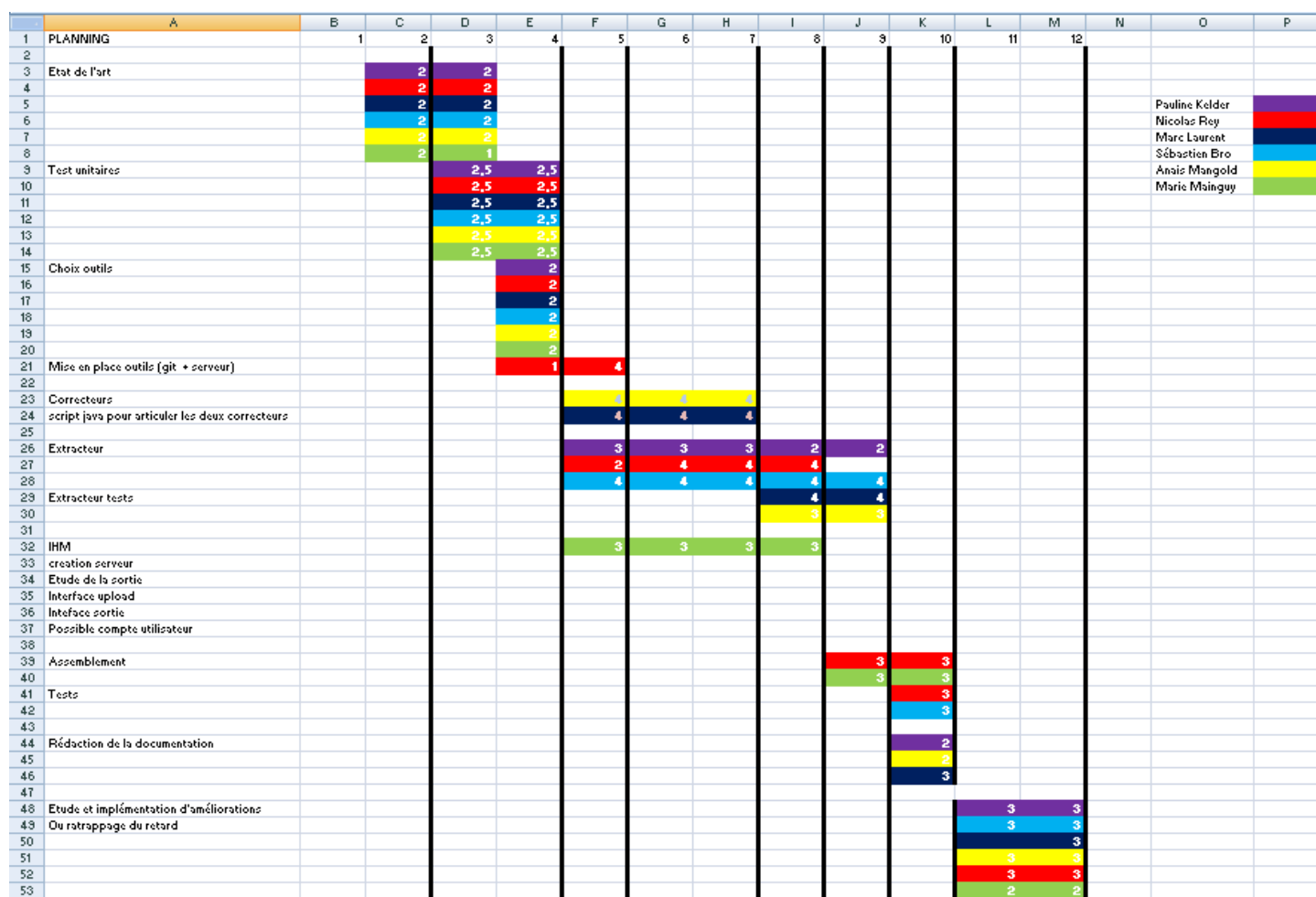


Figure 3 - Planning 1 - datant du milestone : Rendu cahier des charges

Toutefois nous nous sommes rendu compte que relier les différents outils était très rapide contrairement à nos prévisions et que l'interface homme machine étant en JEE nécessitait bien plus de travail (notamment apprendre le JEE !)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	PLANNING HORAIRE	1	2	3	4	5	6	7	8	9	10	11	12			
2																
3	Etat de l'art		2	2												
4			2	2												
5			2	2												
6			2	2												
7			2	2												
8			2	1												
9	Test unitaires			2,5	2,5											
10				2,5	2,5											
11				2,5	2,5											
12				2,5	2,5											
13				2,5	2,5											
14				2,5	2,5											
15	Choix outils				2											
16					2											
17					2											
18					2											
19					2											
20					2											
21	Mise en place outils (git + serveur)				1	4	4	4	4	4						
22																
23	Correcteurs					4	4	4	4							
24	script java pour articuler les deux correcteurs					4	4	4	4							
25																
26	Extracteur					3	3	3	3	2	2					
27																
28	tests								4	4						
29									3	3						
30																
31	IHM					3	3	3	3							
32	IHM					3	3	3	3							
33	creation serveur															
34	Etude de la sortie															
35	Interface upload															
36	Interface sortie															
37	Possible compte utilisateur															
38																
39	Assemblage									3	3					
40										3	3					
41	Tests											3	3			
42												3	3			
43																
44	Rédaction de la documentation											2	2			
45												2	2			
46												3	3			
47																
48	Etude et implémentation d'améliorations											3	3			
49	Ou rattrapage du retard											3	3			
50												3	3			
51												3	3			
52												3	3			
53												2	2			

Figure 4 - Planning 2 - datant du milestone : choix définitif des outils

Enfin cette répartition fut abandonnée afin de favoriser le travail par binôme. De plus, certaines activités étaient sous évaluées : notamment la réflexion sur l'architecture des classes ainsi que la liaison IHM / Traitement du fichier.

Par ailleurs, certaines tâches furent ajoutées : création de la sortie HTML, par exemple.

Voici donc le planning final qui fut appliqué et respecté jusqu'à la fin du projet.

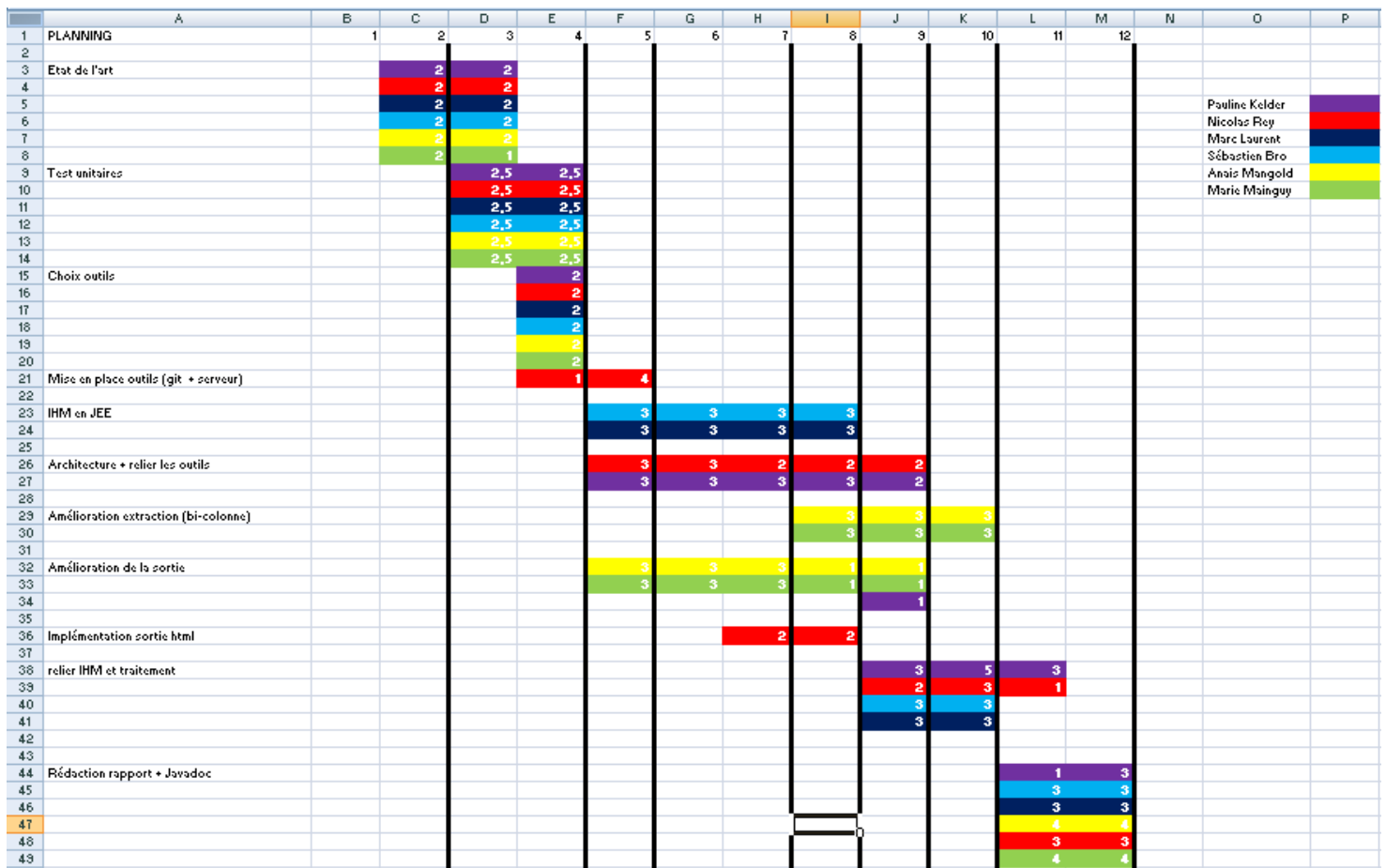


Figure 5 - Planning 3 - datant du milestone : fin de mise en place des outils (git)

Répartition du travail (choix de travail par binôme)

La majorité des tâches fut effectuée par binôme, que ce soit pour l'état de l'art ou le développement de chaque partie de l'application. Cette répartition permet de couvrir de nombreuses tâches en parallèle tout en exploitant la diversité de nos machines quant au développement. Concernant l'état de l'art, ce fut un moyen de tester rapidement un maximum d'outils.

Nous avons essayé de répartir le développement de l'application selon les capacités ou les aspirations de chacun :

- Interface Web : Marc Laurent et Sébastien Bro dans le but d'apprendre le JEE
- Architecture, liaison des outils : Pauline Kelder et Nicolas Rey
- Amélioration de la sortie et de l'extraction : Marie Mainguy et Anaïs Mangold

Nicolas ayant le seul ordinateur sous GNU Linux, il fut responsable de l'intégration continue du code et de la gestion de Hunspell.

		2	4,5	4,5	3	3	3	2	2	2	3	3																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							</
--	--	---	-----	-----	---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Nous avons planifié 287 heures de travail lors de la rédaction du cahier des charges (cela correspond au premier planning). Ce fut en réalité 320 heures.

Cela s'explique par une sous estimation du temps pour faire le lien entre l'interface et l'outil de traitement.

Management

Nous avons commencé par définir des rôles clairs pour chacun, nous les avons écrits dans le cahier des charges puis sur Podio.

Chef de Projet:	Marie MAINGUY
Responsable Communication Client:	Anaïs MANGOLD
Responsable Technique:	Nicolas REY
Responsable Reporting:	Pauline KELDER
Equipe projet:	Anaïs MANGOLD, Nicolas REY, Pauline KELDER, Marc LAURENT, Sébastien BRO, Marie MAINGUY

Afin de suivre les avancées des uns et des autres et pour demander éventuellement de l'aide aux autres membres du groupe, nous avons fait des réunions hebdomadaires le lundi midi. C'est ainsi que nous avons prévenu tout retard et validé l'avancement du projet.

Nous avons, de plus, fait trois réunions avec nos tuteurs afin de les tenir au courant de notre avancement en plus de nos rapports hebdomadaires du vendredi. Lors de ces réunions, nous explicitons le travail effectué, les problèmes rencontrés, les futures tâches à réaliser, ainsi que la nouvelle répartition des tâches.

Podio

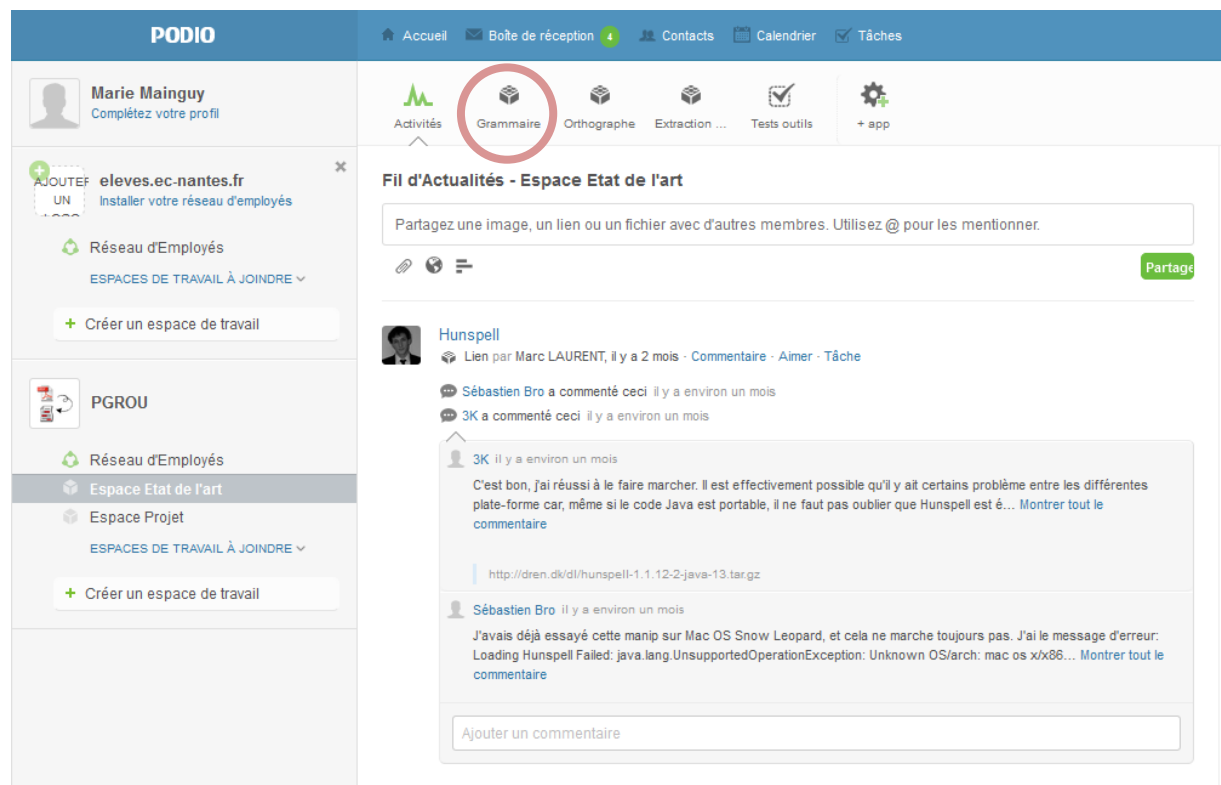


Figure 6 - Espace état de l'art avec les différentes applications déployées

Podio est une plateforme web pour le suivi de projet. On peut y créer les applications nécessaires en simple cliquer-glisser, y partager des fichiers, commenter, s'entre-aider... Elle nous a surtout servi lors de l'état de l'art. Nous avons créé des applications pour chaque thème de notre état de l'art : Correcteur orthographique, Correcteur grammatical et extracteur de texte pour document PDF. Afin de faire notre choix, nous avons testé chaque outil. Grâce à Podio, nous avons créé un formulaire de test afin de pouvoir comparer par la suite les outils. En cas de problème pour tester un outil, nous y posons nos questions... Cela a très bien fonctionné.

De plus, le planning avait été rentré dans Podio sous forme de Milestones. L'application nous rappelait en avance lorsque nous arrivions à un moment charnière du projet. De plus, nous avons désigné des responsables pour chacun.

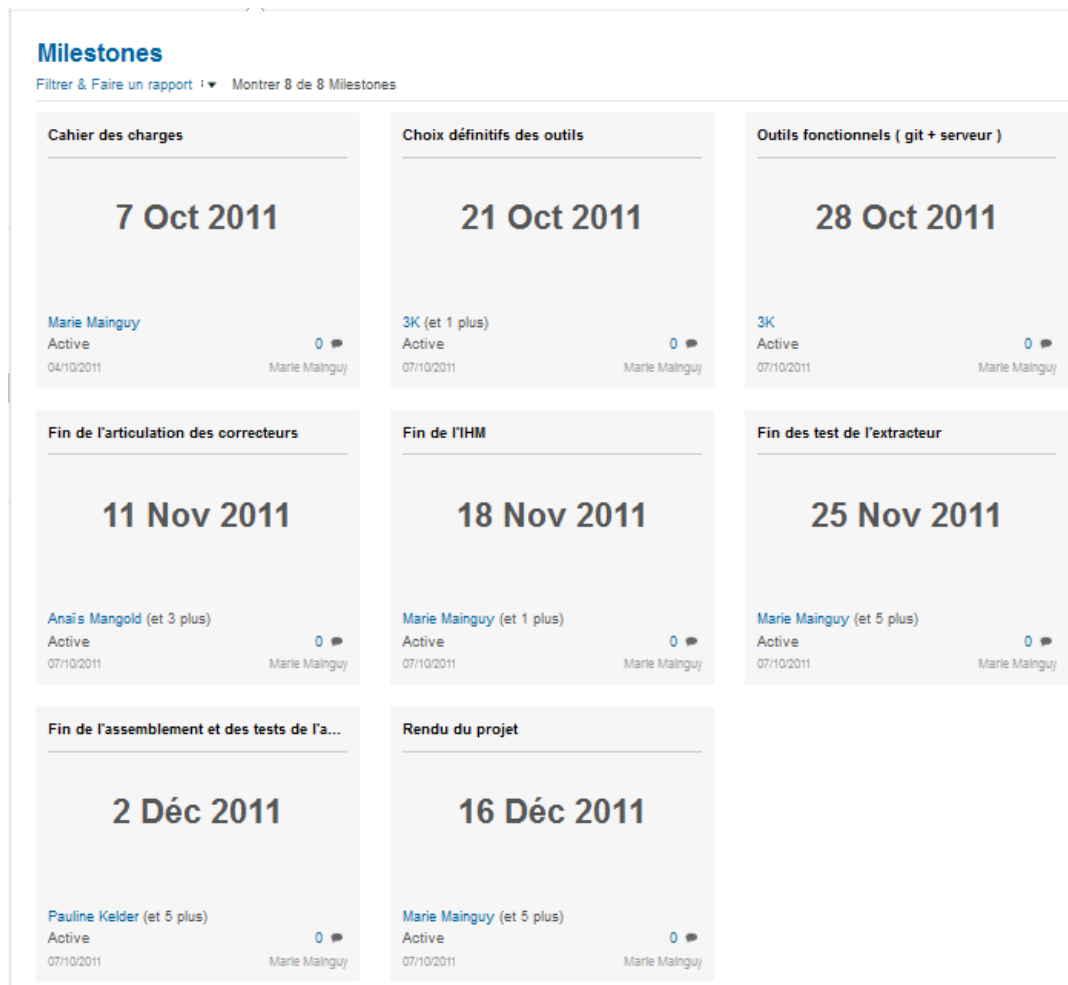


Figure 7 - Application milestones

Lorsque nous sommes arrivés dans la phase de développement, Podio n'a plus servi qu'à planifier nos réunions et le code fut partagé sur GitHub.

Dépôt git

Mis en place par Nicolas, le dépôt Git permet à tout le monde d'avoir accès aux fichiers à jour ainsi que de partager leur travail. Nicolas vérifiait chaque code avant de le publier. Ce dépôt nous permit aussi de donner un accès à notre travail à nos tuteurs. Ils purent ainsi suivre nos avancées tout en nous guidant (notamment concernant la modularité du code).

III. Déroulement du projet

Nous allons reprendre le déroulement du projet étape par étape et détailler pour chacune de ces étapes les éléments suivants : durée de réalisation, difficultés rencontrées, choix de réalisation, organisation du travail

Etat de l'art

Durant les trois premières semaines du projet, nous avons réalisé un état de l'art concernant les outils d'extraction de texte PDF, de correction grammaticale et orthographique. Ces outils ont ensuite été testés. Ces trois états de l'art et les phases de test ont été réalisés en binômes sur chaque outil. Voici trois tableaux récapitulatifs des résultats des états de l'art menés pour chaque outil :

EXTRACTEUR DE PDF

Nom de l'outil	Langage utilisé	Simple d'utilisation ?	Updates disponibles ?
pdftotext	Java	<ul style="list-style-type: none">• Outil GNU/Linux installé par défaut sur la plupart des distributions• Gère assez bien les colonnes• Non-gestion des exposants, des en-têtes et des pieds-de-page qu'il ne différencie pas du reste du texte. Les accents posent aussi problème.• Supprime automatiquement les tirets lorsqu'un mot est coupé à la fin d'une ligne.• Très simple d'utilisation	Oui
PDFBox	Java	<ul style="list-style-type: none">• Non-gestion des exposants, des accents et des en-têtes et pieds de page.• Les mots coupés par un tiret "-" ne sont pas recomposés, le passage à la ligne est immédiat.• Les colonnes sont bien gérées.	Oui
Tika	Java	Librairie très grosse contenant notamment PDFBox	Oui
PDF Text Extraction for Java	PAYANT		
jPDFText	PAYANT		

Lors de notre recherche des outils, nous avons expérimenté "pdftotext" et "PDFBox". Le premier est en fait une commande Unix permettant d'extraire le texte d'un PDF, alors que le deuxième est une bibliothèque Java. Bien que "pdftotext" soit utilisable depuis Java en faisant des appels du Runtime, cela posait tout de même des problèmes de portabilité et rendait l'utilisation compliquée. Dans la mesure où les deux extracteurs ont globalement les mêmes performances, nous nous sommes orientés vers PDFBox du fait qu'il soit directement utilisable en Java.

PDFBox permet, comme on l'a dit, d'extraire du texte depuis un PDF. Cependant, il n'est pas parfait car il ne fait pas, par exemple, la différence entre un exposant et un caractère de la ligne. Les en-têtes, les pieds de pages, les numéros de pages, et les légendes des images sont entre autre vus comme le contenu du document, et donc extraits comme tels. Le texte résultant de l'extraction peut donc se trouver pollué par ces caractères insérés au milieu du texte. Cela complique la correction puisque les correcteurs détectent alors des faux positifs.

CORRECTEUR ORTHOGRAPHIQUE

Nom de l'outil	Langage utilisé	Simple d'utilisation ?	Updates disponibles ?
GNU Aspell	C ++	<ul style="list-style-type: none"> Ancienne version d'Hunspell 	Oui
Hunspell	C ++	<ul style="list-style-type: none"> Utilisé sous Mozilla Firefox, Thunderbird, Chrome, MAC OSX, Opera, OpenOffice Nous ne sommes pas parvenus à le faire fonctionner sous Mac et sous Windows mais fonctionne sous Linux 	Oui

Nous avons ensuite choisi d'utiliser Hunspell, correcteur orthographique ayant fait ses preuves puisqu'utilisé par Open Office et Firefox (entre autres).

CORRECTEUR GRAMMATICAL

Nom de l'outil	Langage utilisé	Simple d'utilisation ?	Updates disponibles ?
LanguageTools	Java	<ul style="list-style-type: none"> Très facile d'utilisation en ligne de commande (moins de 2 minutes pour le tester). Renvoie la liste des erreurs grammaticales et leur ligne avec une proposition de correction. 	Oui
Lightproof	Python	<ul style="list-style-type: none"> Extension d'OpenOffice. Nous n'avons pas réussi à le faire fonctionner en ligne de commande. 	Non

LanguageTools fut choisi tout d'abord pour son développement en Java (Lightproof étant en python, il était plus difficile de l'intégrer au projet). Il s'inclut comme librairie dans notre projet. Il fut donc particulièrement simple à tester en ligne de commande puis à intégrer dans Eclipse.

Disponible en plusieurs langues, il sera possible par la suite de choisir la langue de notre PDF (il existe des dictionnaires en anglais, français, allemand, polonais... etc), la version anglaise restant toutefois la plus complète. Il teste aussi bien les erreurs de grammaire que les erreurs de style (par exemple : un espace seulement après la virgule et non un devant et derrière).

La documentation étant complète et très accessible, nous avons pu rapidement l'intégrer à notre projet, un exemple de création d'objet JLanguageTool est donné sur le site. Ce fut notre point de départ pour l'écriture de notre script.

DIFFICULTES RENCONTREES

Même si à long terme la diversité de nos systèmes d'exploitation au sein du groupe (2 Windows Seven, 3 Mac OSX, 1 GNU Linux) a constitué une force lors des phases finales de test de l'outil que nous avons réalisées, durant cette première phase de test des outils mis à l'étude, cela a constitué une difficulté majeure pour le test de certains outils (notamment les correcteurs orthographiques).

Architecture et articulation des outils

Nous avons divisé le moteur de l'application en trois parties indépendantes : l'extraction du texte, la correction grammaticale, et la correction orthographique. Les exigences du client étaient d'avoir une structure modulaire et évolutive afin que des projets futurs puissent continuer sur la base de notre travail. Nous avons donc pris soin de pouvoir facilement changer les outils utilisés. Ainsi, les correcteurs grammaticaux et orthographique héritent chacun des classes abstraites GrammaticalCorrection et OrthographiqueCorrection qui elles-mêmes implémentent l'interface Correction qui permet d'imposer les méthodes utilisées lors du traitement des corrections. La conception est équivalente pour l'extracteur de texte.

Le traitement de la correction se fait dans une classe : CorrectionResult. C'est cette classe qui est chargée de récupérer les erreurs trouvées par les correcteurs et de les mettre en forme. Nous avons choisi deux types de mises en forme : une forme HTML destinée à être affichée dans le navigateur de l'utilisateur afin qu'il choisisse parmi les propositions de correction, et une autre qui consiste en une liste détaillée de toutes les erreurs trouvées dans le document.

Diagramme de classes

Amélioration de l'extraction

Deux axes d'amélioration ont été menés :

- Supprimer les en-têtes et pieds de page lors de l'extraction (éviter l'apparition au milieu du texte du numéro de page, de l'en-tête, des notes de bas de page)
- Gérer le bi-colonnes et si possible le multi-colonnes.

L'amélioration a été directement intégrée à la classe Extraction.java réalisant l'extraction du PDF, au sein même de la méthode textExtraction().

SUPPRESSION DES EN-TETES ET PIEDS DE PAGES

Après quelques recherches au sein de la bibliothèque pdfbox, il s'est avéré que des classes java permettant la sélection de rectangles de texte existaient déjà. On a donc procédé à l'importation des bibliothèques suivantes :

```
import org.apache.pdfbox.util.PDFTextStripper;
import org.apache.pdfbox.util.PDFTextStripperByArea;
import java.awt.Rectangle;
```

Dans un premier temps, nous avons cherché à simplement sélectionner un rectangle de texte page par page, excluant l'en-tête et le pied de page.

Pour cela nous définissons une marge en haut et une marge en bas, nous enlevons ces marges à la taille de la page obtenue grâce à :

```
int x = (int) document.getPageFormat(0).getWidth();
int y = (int) document.getPageFormat(0).getHeight();
```

Malheureusement getPageFormat() est une fonction dépréciée mais nous n'avons pas trouvé de substitut. Cela sera sûrement à revoir dans les versions futures de l'application.

GESTION DU MULTI-COLONNES

Dans un second temps, nous avons cherché à pouvoir traiter un nombre de colonnes donné page par page. On définit donc au début de la classe Extraction.java, l'entier nbCol qui représente le nombre de colonnes par page du fichier PDF. L'utilisateur pourra notamment renseigner cette variable sur l'interface web.

Nous allons détailler l'algorithme, ce dernier explicitera aussi la prise en compte des bas et hauts de page.

Algorithme

```
Entier nbColonnes ;
Entier margeHaut ;
Entier margeBas ;
PDFDocument docPdf ;
String texteExtrait ;

Liste toutesLesPages = docPDF.récupérerLesPages() ;
Entier nbPage = toutesLesPages.taille() ;

Pour entier i de 0 à nbPage{
    PDFPage pageCourante = toutesLesPages.atteindre(i) ;
    Entier x = pageCourante.largeur() ;
    Entier y = pageCourante.hauteur() ;

    Pour entier j de 0 à ( nbColonnes-1){
        Rectangle zone = nouveau Rectangle
        ( 0+j*x/nbColonnes, margeHaut, x/nbColonne, y-margeHaut-margeBas) ;
        // (XcoinGauche, YcoinGauche, largeur, hauteur)
        texteExtrait=texteExtrait ^pageCourante.extraire(zone) ;
    }
}
```

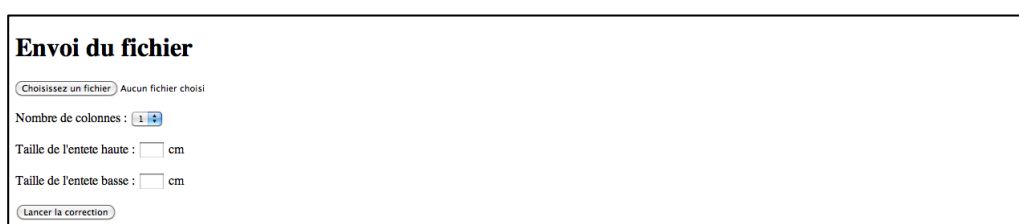
Réalisation de l'interface web

Afin de faciliter l'utilisation de notre outil pour l'utilisateur, nous avons créé une interface web qui lui permet de corriger son document à partir de son navigateur, sans avoir à installer un quelconque utilitaire. Etant donné notre décision d'utiliser le langage Java dans notre programme, nous avons donc développé cette interface en JEE, pour Java Enterprise Edition, et d'héberger notre projet sur un serveur Java.

PRINCIPE

L'utilisateur accède à la page d'accueil, sur laquelle il peut uploader un rapport en version PDF, en mentionnant le nombre de colonnes qu'il contient, ainsi que la taille des marges hautes et basses.

Il peut ensuite lancer la correction, qui commence par uploader le fichier sur le serveur, puis en extrait le texte brut à partir des indications rentrées dans le formulaire, et fait appel aux fonctions orthographique et grammaticale pour renvoyer un fichier texte contenant la liste des erreurs détectées accompagnées de suggestions de corrections. L'utilisateur est alors envoyé à une page de résultats, sur laquelle il pourra au choix télécharger ce fichier, ou bien procéder à une correction directement sur son navigateur « Word-style » en précisant les corrections qu'il souhaite faire, pour enfin télécharger son texte corrigé sous format TXT.



Envoi du fichier

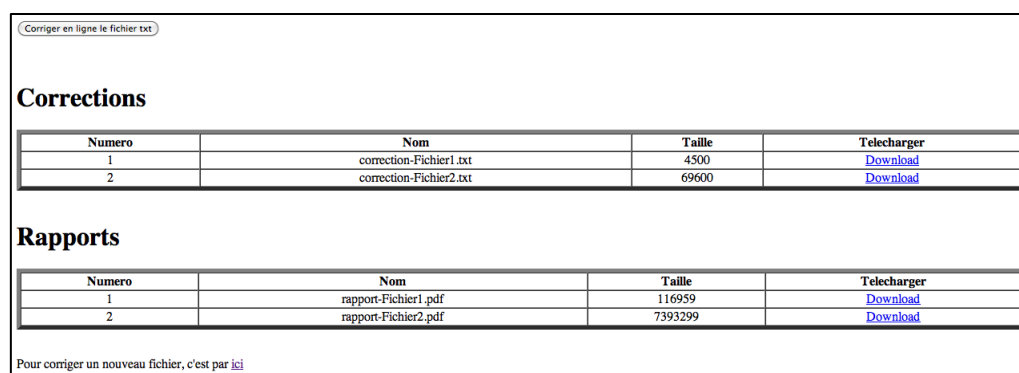
Aucun fichier choisi

Nombre de colonnes :

Taille de l'entete haute : cm

Taille de l'entete basse : cm

Figure 8 - Aperçu de l'interface d'envoi du fichier



Corrections

Numero	Nom	Taille	Telecharger
1	correction-Fichier1.txt	4500	Download
2	correction-Fichier2.txt	69600	Download

Rapports

Numero	Nom	Taille	Telecharger
1	rapport-Fichier1.pdf	116959	Download
2	rapport-Fichier2.pdf	7393299	Download

Pour corriger un nouveau fichier, c'est par [ici](#)

Figure 8 bis - Aperçu de l'interface d'envoi du fichier

FONCTIONNEMENT

Pour créer cette interface, nous nous sommes basé sur un code proposé par Taher Tinwala, qui permet d'uploader et de downloader un fichier sur un serveur Java. Ce code utilise deux bibliothèques, commons-io et commons-fileupload, développés par le projet Apache Commons. La première s'occupe de la gestion des fichiers d'entrée/sortie, tandis que la seconde permet l'upload de fichiers vers un serveur Java.

Le fichier **web.xml**, qui s'occupe du mapping des servlets, va uniquement servir à lier la racine du site à notre servlet, **FileUpload.java**, qui s'occupe de gérer les requêtes d'affichage, d'upload, de download et de correction du rapport. Au chargement du site, il invoque **Index.jsp**, qui contient le code HTML comprenant le formulaire d'upload. Lors de la demande d'upload d'un fichier, notre servlet va stocker celui-ci sur le serveur, par *méthode directe* pour les petits fichiers ou par *méthode streaming* pour les autres (les tailles limites peuvent être choisies dans le servlet). Il va ensuite appeler le fichier **Main.java**, qui va extraire puis corriger grammaticalement et orthographiquement le texte du rapport, en utilisant les fonctions contenus dans le package *com.pgrou.pdfcorrection*, pour enfin générer une liste d'erreurs et de suggestions de correction. Le servlet envoie ensuite l'utilisateur sur la page **result.jsp**, sur laquelle il va trouver l'ensemble des fichiers uploadés ainsi que les fichiers d'erreurs et de suggestions correspondants. Ces éléments sont tous rangés dans le même dossier, et la classe **Files.java** permet de les sélectionner suivant leur extension, et donc leur type (rapport/correction). Enfin, l'utilisateur peut corriger son texte à partir des suggestions directement sur son navigateur, à l'instar des éditeurs de texte classiques, en faisant appel au fichier **correction_html.jsp** : en cliquant sur une erreur, l'utilisateur peut choisir de la remplacer parmi une liste de propositions.

Le schéma suivant permet de visualiser le fonctionnement global de l'interface.

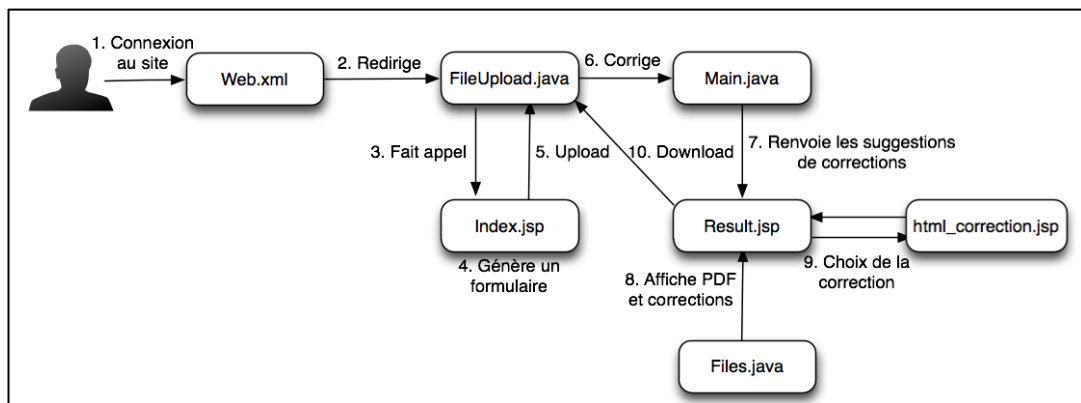


Figure 9 - Diagramme de collaboration

Relier l'outil à l'interface

Une fois l'ensemble des éléments mis en place séparément, la phase finale du projet fut de relier l'interface web à l'outil de correction réalisé en java.

Ce lien est fait à l'aide des classes **FileUpload.java** et **Main.java**. Celles-ci sont en effet sollicitées par les deux parties. Dans un premier temps, la requête d'upload lancée par l'utilisateur est transmise à **FileUpload.java**. Celle-ci permet l'ajout du fichier pdf au dossier "Correction" et lui permet d'être téléchargeable à tout moment. Puis, une fois le fichier pdf récupéré, un appel à la méthode `correction_final` présente dans la classe **Main** est lancé. C'est cette méthode qui déclenche la correction et articule les trois outils de correction: extraction, correction grammaticale, et correction orthographique.

On peut ainsi représenter le lien outil interface de la façon suivante:



Améliorations diverses apportées à l'outil

EXTRACTION DU TEXTE

Dans la classe PDFExtraction, on définit un attribut `String[] patternToRemove` ainsi qu'une méthode `cleanText()` qui enlève du texte extrait tous les caractères situés dans l'attribut `patternToRemove`. Cela permet donc de gérer les caractères non désirés du type `"#!^$()[]{}?+*.\\|"`.

CORRECTION ORTHOGRAPHIQUE:

Dans la classe HunspellOrthographicCorrection.java qui spécifie les méthodes de correction orthographique du texte, et plus particulièrement dans la méthode `correctText()`, on place les mots mal orthographiés mais également les mots **ne contenant pas** des caractères tels que `#, $, +, *` dans un `HashMap` ayant pour clé la liste de mots avec les suggestions de corrections associées. Ainsi on exclut les mots contenant les caractères `"#!^$()[]{}?+*.\\|"`.

IV. Résultat final : présentation de l'outil

Écart avec le cahier des charges

Peu au courant des technologies à notre disposition lors de la rédaction du cahier des charges, nous nous étions engagés au minimum au traitement de fichiers PDF simples, soit des fichiers mono colonne, sans en-tête ni bas de page.

Nous avons donc étendu notre projet à la prise en charge de fichiers PDF plus complexes :

- ✓ Avec en-tête et bas de page
- ✓ Avec plusieurs colonnes

Mais toutefois avec les limites suivantes :

- Colonne de même taille.
- Soit mono-colonne soit multi-colonne mais pas de mélange des deux sur la même page.
- Pas de prise en compte des images
- Légendes et annotation prises en compte au milieu du texte.

Concernant la sortie du correcteur, notre engagement minimal était un fichier txt contenant la liste des erreurs. Il est en effet implémenté.

Nous avons ajouté une fonctionnalité permettant de corriger le texte en ligne et de récupérer le texte corrigé en txt. C'est un fichier jsp qui affiche les mots à corriger en rouge et qui propose au clic les différentes solutions proposées par le correcteur. Il n'est toutefois pas possible d'écrire soi-même un mot qui n'est pas proposé.

Concernant l'interface web permettant de soumettre un fichier PDF, nous avons fait ce à quoi nous nous étions engagés : une page web avec un champ uploader le PDF. Nous avons rajouté la possibilité de spécifier les marges et la taille des en-têtes.

Le cahier des charges est donc parfaitement respecté.

Diagrammes descriptifs de l'outil final

DIAGRAMME DE CAS D'UTILISATION

Le diagramme de cas d'utilisation présente l'outil dans son ensemble.

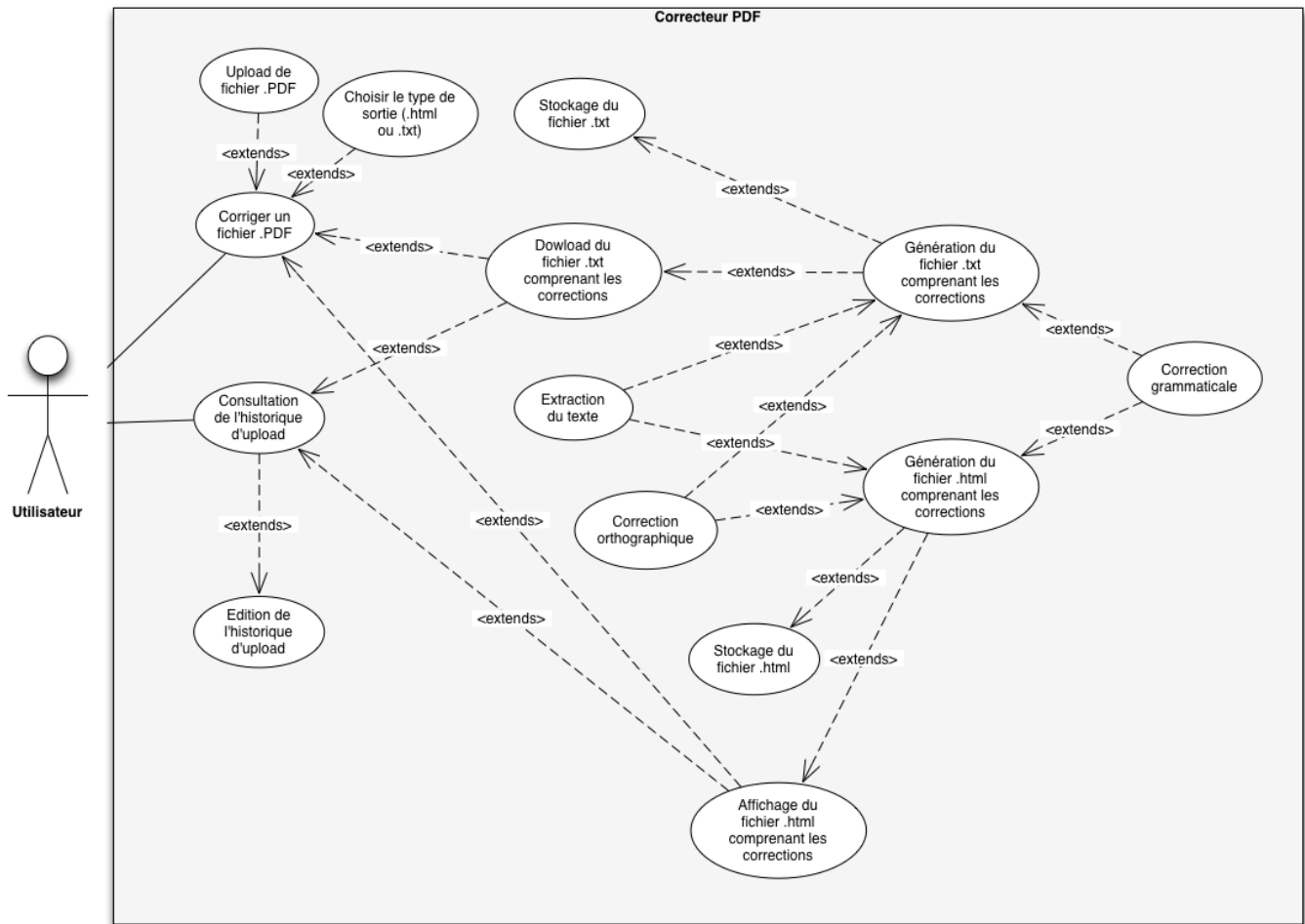


Figure 10 - Cas d'utilisation

Identification

Nom du cas : Corriger un PDF

But : détaille les étapes permettant à un utilisateur de corriger un fichier .PDF

Acteur principal : Application PDF Corrector

Acteur secondaire : Utilisateur

Date : le 21/11/11

Responsable : Mlle MAINGUY

Version : 1.0

Séquencement

Le cas d'utilisation commence lorsqu'un utilisateur upload un PDF.

Enchaînement nominal

L'utilisateur sélectionne un fichier à uploader.

L'application valide le format du fichier et effectue l'upload.

L'utilisateur choisit le type de sortie qu'il désire (.html ou .txt).

L'application extrait le texte du PDF.

L'application stocke le texte brut.
 L'application corrige la grammaire de ce texte brut.
 L'application corrige l'orthographe de ce texte brut.
 L'application enregistre les corrections suggérées dans un fichier .txt
 L'application génère le .html si tel était le souhait de l'utilisateur.
 L'application affiche la page .html à l'utilisateur.
 L'utilisateur télécharge le fichier .txt si c'est le type qu'il a choisit précédemment.

DIAGRAMME DE SEQUENCES

Que se passe-t-il lorsqu'un PDF est envoyé à l'application ?

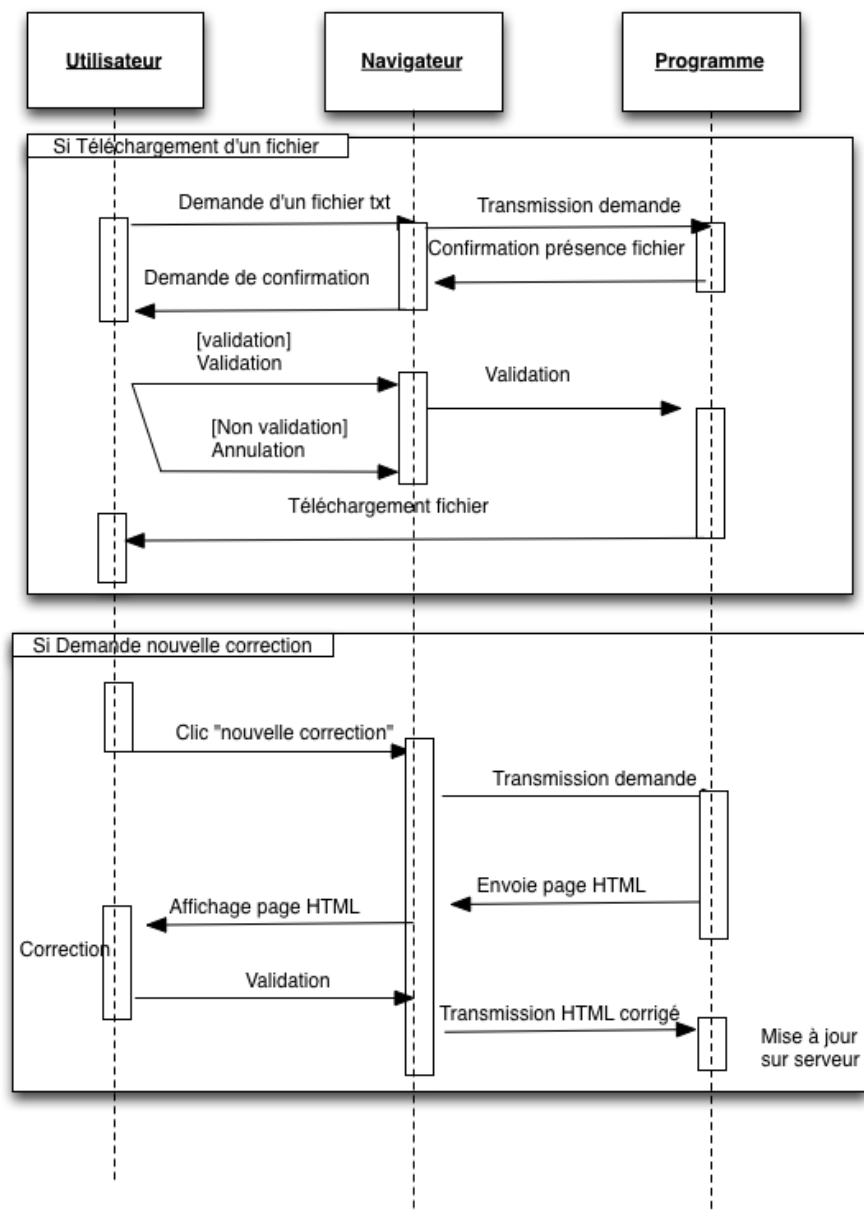


Figure 11 - Diagramme de séquence

DIAGRAMME COMPOSANTS

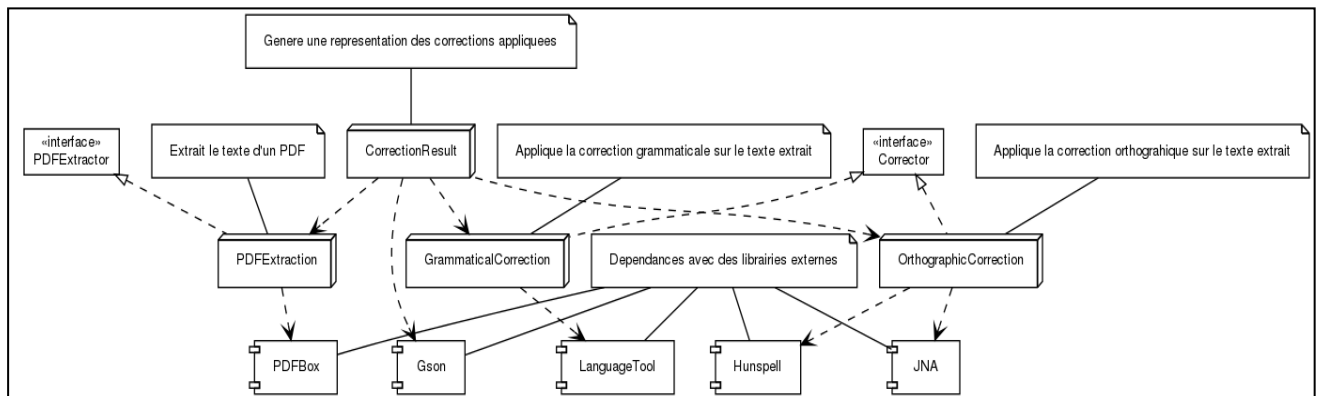


Figure 12 - Diagramme de composants

DIAGRAMME DE DEPLOIEMENT

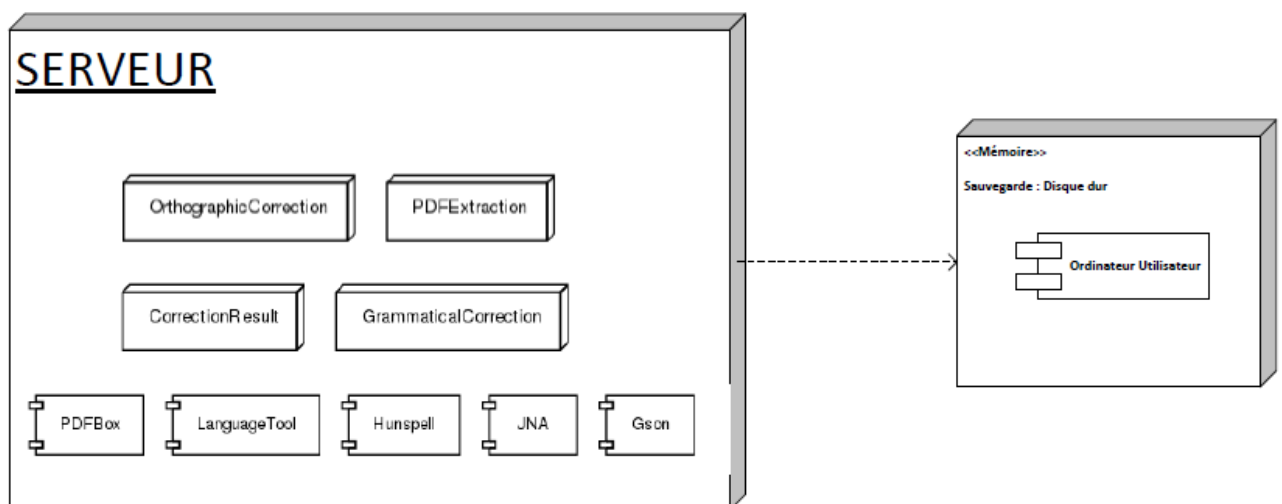


Figure 13 - Diagramme de déploiement

Modularité du code

Comme présenté précédemment (voir Partie II, Architecture et articulation des outils), le point fort de cette application est sa grande modularité. En effet, nous avons pris soin, à la demande de notre client, de garantir une flexibilité importante. Chacun des trois outils utilisés lors de la correction du fichier PDF peut ainsi être facilement changé au besoin sans pour autant affecter l'ensemble du fonctionnement de PDFCorrector.

Plus concrètement, chaque outil hérite d'une classe abstraite définissant sa fonction : `PDFSimpleExtraction`, `OrthographicCorrection` et `GrammaticalCorrection`. Ce sont les méthodes de ces classes abstraites qui sont appelées dans la méthode principale `correction_final`. Ainsi, si on choisit de changer d'outil de correction grammaticale par exemple, il suffit de créer une classe de correction utilisant cet outil et héritant de

GrammaticalCorrection. Une fois la méthode correctText redéfinie dans la nouvelle classe, il s'agit simplement de préciser dans la class Main le nom de l'outil à utiliser lors de l'instanciation de l'objet de correction grammatical utilisé.

Actuellement, les trois outils choisis sont définis par les classes : PDFBoxExtractor, LanguageToolGrammaticalCorrection, HunspellOrthographicCorrection (ces noms sont certes long mais très explicites).

Licence

L'application PDF Corrector est publié sous la licence LGPL soit la licence GPL amoindri.

Voir sur : <http://www.gnu.org/licenses/lgpl.html>

En effet aucun des outils utilisés n'est sous licence GPL, nous sommes donc affranchis du copyleft.

V. Pistes d'améliorations envisagées

Sélectionner les zones à corriger

PRINCIPE

Les PDF sont rarement composés de colonnes ayant exactement la même taille. Afin d'avoir un texte cohérent malgré une mise en page complexe, il serait intéressant de pouvoir sélectionner soi-même les zones du PDF à corriger. Il faudrait donc pouvoir visualiser en ligne le PDF et effectuer manuellement, à l'aide de la souris, la saisie du rectangle.

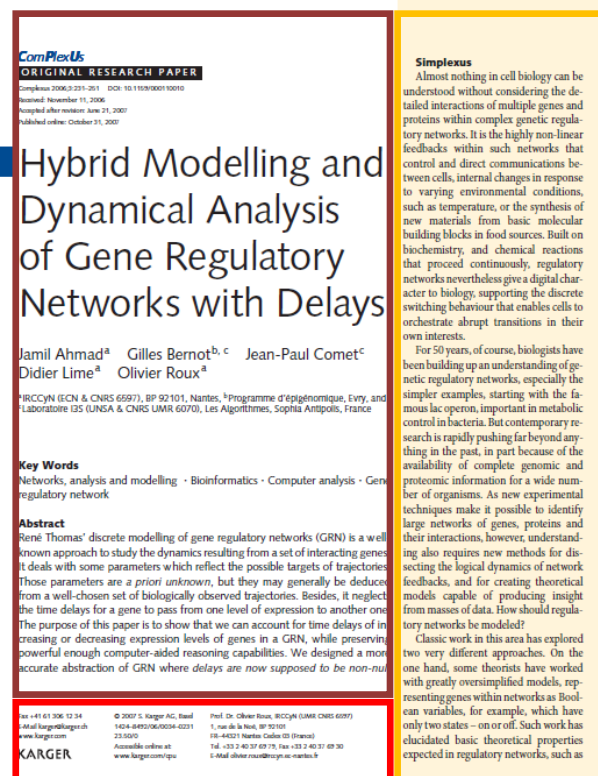


Figure 14 - Sélection des zones à corriger

PISTES TECHNOLOGIQUES

Grâce au CSS3, HTML5 et JavaScript, il serait possible de récupérer la taille des rectangles ainsi que leur emplacement sur la page. Quand à la création de l'image, il existe une méthode PDFTolImage qui est en bêta. Elle pourra servir pour créer le support à la sélection en créant une image pour chaque nouvelle page du PDF (Source : <http://pdfbox.apache.org/commandlineutilities/PDFTolImage.html>).

Des comptes utilisateurs avec un historique

PRINCIPE

Le principe est très simple. Il s'agirait de proposer la création d'un compte à l'utilisateur : il pourrait ainsi y sauvegarder différentes versions de correction d'un même PDF avec un historique de toutes les corrections réalisées et la possibilité de récupérer chacune d'elles. L'interface web pourrait être ainsi dotée d'un service de versionning

Organisation du répertoire d'une session utilisateur

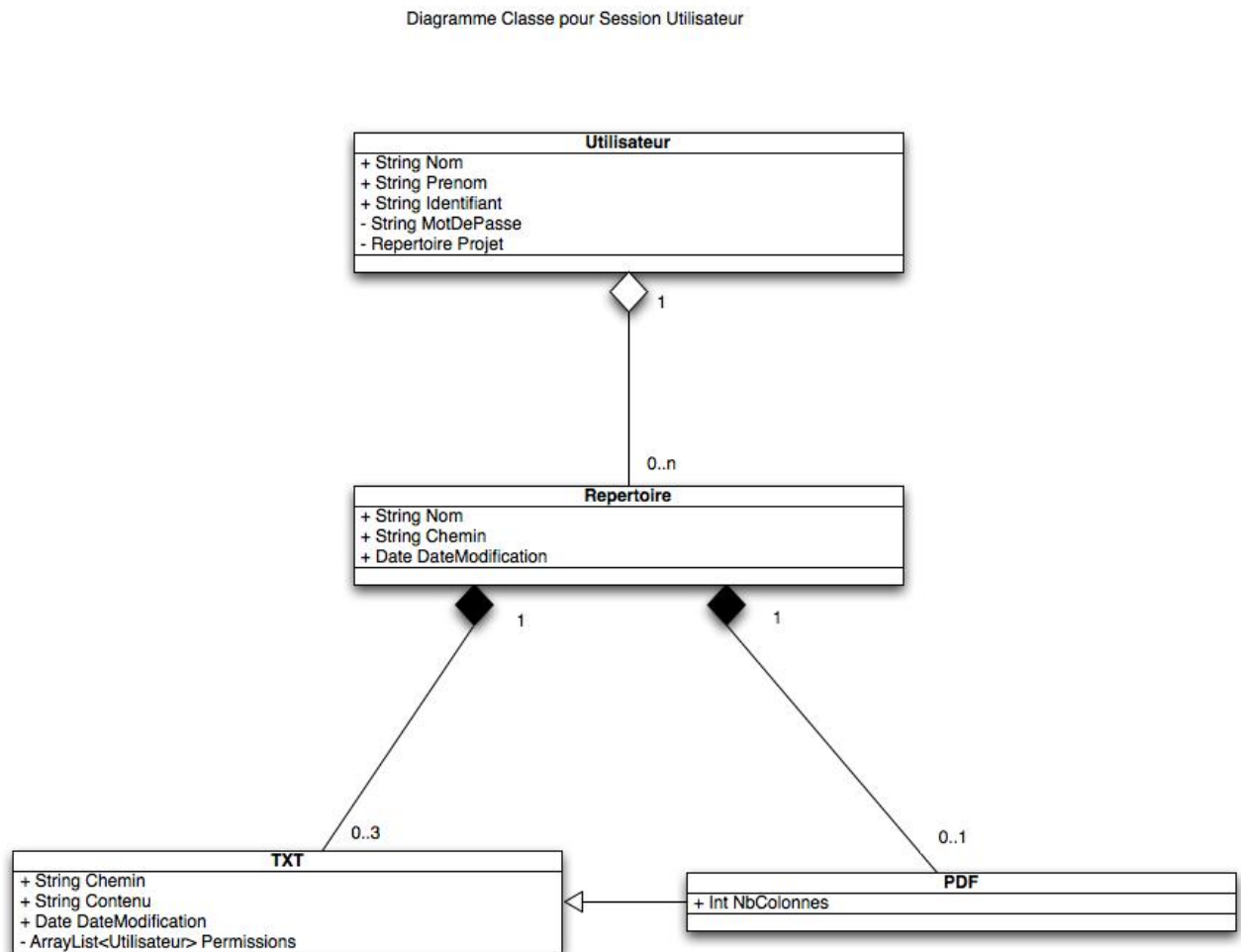


Figure 15 - Diagramme de classe pour une session utilisateur

Diagramme Objet pour Session Utilisateur

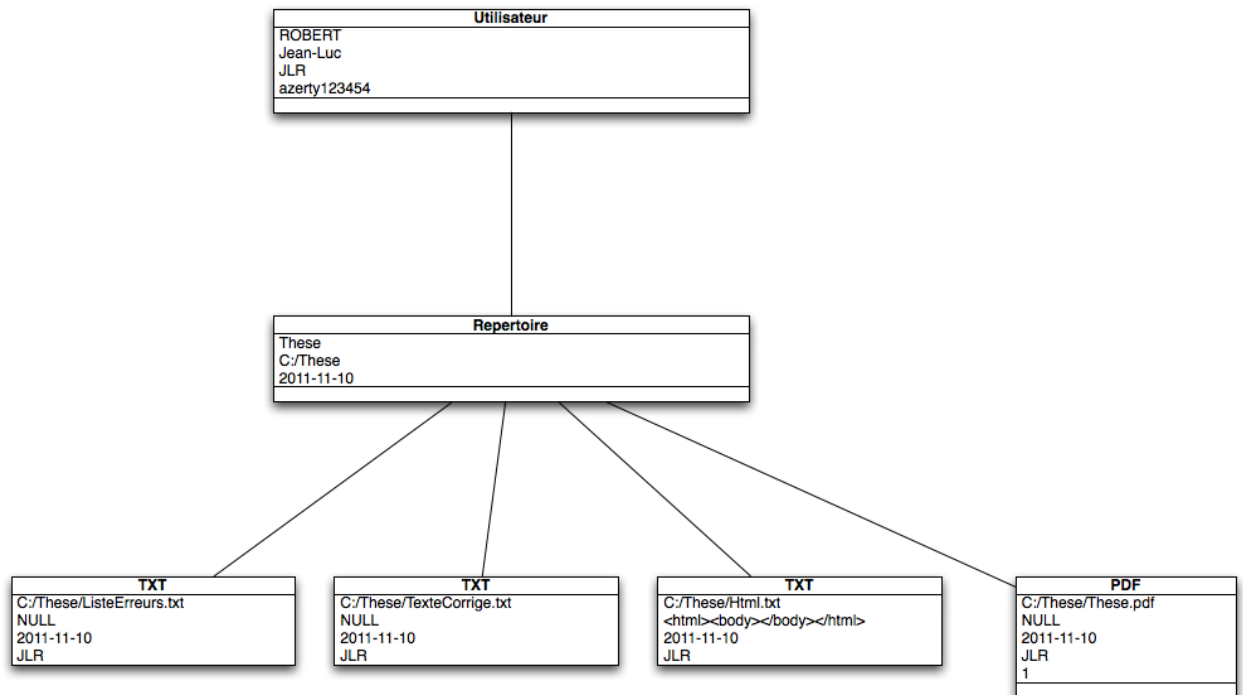


Figure 16 - Diagramme objet pour une session utilisateur

PISTES TECHNOLOGIQUES

En implémentant n'importe quelle base de données compatible avec java, on pourrait ainsi gérer toutes les informations et données liées à un utilisateur et mettre alors en place un historique des corrections de PDF réalisées.

Traitement des formules mathématiques

PRINCIPE

Le but de cette fonctionnalité serait de pouvoir récupérer l'ensemble d'une formule mathématique lors de l'extraction de texte au format PDF. En effet, celles-ci perdent actuellement leur sens (les exposants ne sont pas pris en compte, souvent enregistrées au format d'images, etc.) Dans l'absolu, l'utilisation de variables définies universellement pourrait même permettre une correction de la formule en elle-même (si celle-ci est déjà enregistrée dans une base de données). Mais, nous n'en sommes pas là, il s'agit simplement ici de pouvoir récupérer la formule en tant que telle, et non dénaturée comme c'est le cas actuellement.

PISTES TECHNOLOGIQUES

Malheureusement, pdfBox ne présente pas encore de méthode permettant la récupération d'une formule mathématique. La formule est donc, lorsqu'elle est rencontrée, traitée comme une partie de texte, ce qui

engendre souvent inutilement une alerte d'erreur. Ainsi, cet outil ne permet pas pour l'instant l'ajout d'une telle fonctionnalité. Il s'agirait donc d'envisager une extraction par un autre outil plus développé de ce point de vue-ci.

Amélioration de la correction en ligne après traitement

PRINCIPE

Il serait intéressant de pouvoir sélectionner une correction manuelle si toutes les suggestions ne conviennent pas. Ainsi rajouter un choix « autre... » dans la liste de suggestions et avoir alors la possibilité de retaper le mot.

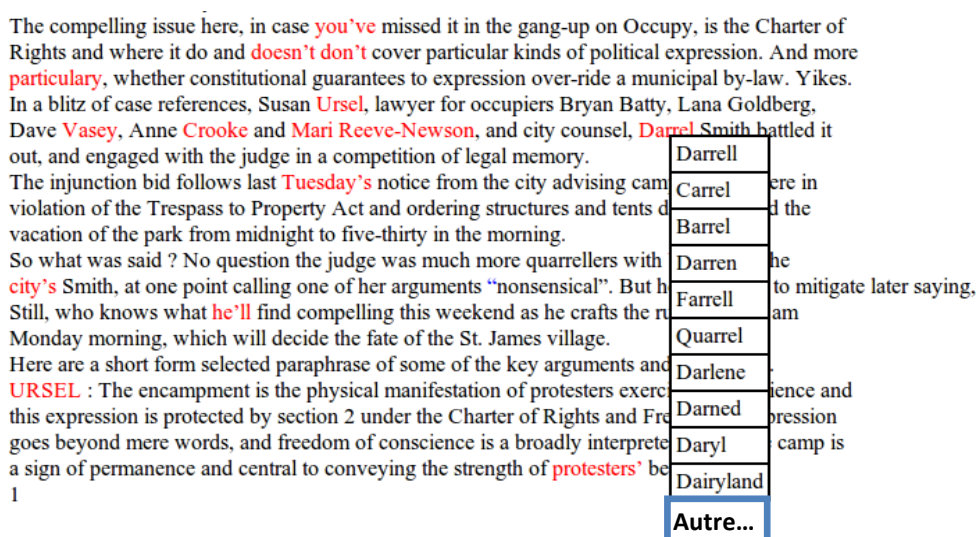


Figure 17 - Amélioration de la correction en ligne

PISTES TECHNOLOGIQUES

Avec un Textarea en html et un peu de JavaScript cela devrait être possible facilement.

Accélération du traitement de la correction.

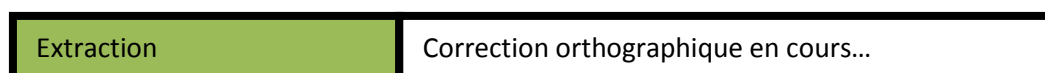
PRINCIPE

Dans l'état actuel, le traitement du PDF est très lent. Le traitement s'effectue en 3 phases : l'extraction du texte, la correction grammaticale, et la correction orthographique. Cette lenteur est due au fait que les algorithmes ne sont pas multi-threadés. On ne tire donc pas avantage des processeurs multi-coeurs qui équipent aujourd'hui la grande majorité des ordinateurs. En ce qui concerne l'extraction, on ne peut pas faire grand chose pour changer ça. En revanche, il est possible de considérablement accélérer les corrections. Dans le cas de la correction grammaticale, on pourrait extraire chaque phrase du texte, et lancer l'analyse de chacune d'elle dans un thread séparé. De la même manière, comme la correction orthographique s'effectue de toute façon mot par mot, chaque mot peut être également analysé dans des threads séparés.

Affichage d'une barre de progresssion

PRINCIPE

Le traitement du PDF prenant du temps, il serait bien de mettre en place une barre de progression synchronisée avec l'avancement du traitement, au lieu d'un simple spinner. L'utilisateur n'aurait ainsi pas de doute quand au bon fonctionnement de l'application.



VI. Conclusion

Les chercheurs écrivent leurs articles sous Latex et les éditent directement en pdf. Pourtant, les fichiers PDF sont des fichiers difficilement et même la plupart du temps pas du tout modifiables. Ils ne peuvent donc vérifier l'orthographe et la grammaire de leurs articles. C'est dans ce cadre que nous a été confié le projet de réalisation d'un correcteur pour PDF.

Après un état de l'art des outils à notre disposition pour réaliser notre projet, nous en avons sélectionné trois : Hunspell, LanguageTools et PDFBox. Nous avons implémenté une interface web en JEE qui permet l'upload du PDF qui est ensuite traité pour enfin un affichage de la liste des erreurs.

Ce projet de groupe s'est déroulé sans problème majeur, la communication fut bonne, les retards moindres et les difficultés techniques surmontées sans encombre. Ce fut une expérience très positive. De plus le projet est voué à évolution, nous avons en effet réfléchi aux différents moyens d'améliorer notre application. Le principal axe d'évolution serait notamment l'amélioration de la zone sélectionnée pour la correction.

Nous voudrions conclure en remerciant nos accompagnateurs Loïc Paulevé et Olivier Roux qui ont été présents et à notre écoute, ainsi que Jean-Yves Martin qui nous a très gentiment aidé pour la partie JEE et sans qui le débuge aurait été bien plus long...

VII. Webographie

<http://pdfbox.apache.org/apidocs/org/apache/pdfbox/examples/util/ExtractTextByArea.html>

<http://pdfbox.apache.org/apidocs/org/apache/pdfbox/pdmodel/PDDocument.html>

<http://www.languagetool.org/development/>

<http://javahunter.wordpress.com/2011/02/25/apache-commons-fileupload-and-download/>

<http://javahunter.wordpress.com/>

<http://commons.apache.org/io/>

<http://commons.apache.org/fileupload/>

<http://commons.apache.org/>

<http://pdfbox.apache.org/commandlineutilities/PDFToImage.html>

<http://hunspell.sourceforge.net/>

<http://www.siteduzero.com/tutoriel-3-112219-apprenez-a-creeer-des-applications-web-dynamiques-avec-jee.html>

<http://www.gnu.org/licenses/lgpl.html>

VIII. Table des figures

Figure 1 - Diagramme de spécification.....	5
Figure 2 - Cycle de vie.....	5
Figure 3 - Planning 1 - datant du milestone : Rendu cahier des charges	6
Figure 4 - Planning 2 - datant du milestone : choix définitif des outils.....	7
Figure 5 - Planning 3 - datant du milestone : fin de mise en place des outils (git)	8
Figure 6 - Espace état de l'art avec les différentes applications déployées.....	10
Figure 7 - Application milestones.....	11
Figure 8 - Aperçu de l'interface	16
Figure 9 - Diagramme de collaboration.....	17
Figure 10 - Cas d'utilisation	20
Figure 11 - Diagramme de séquence.....	21
Figure 12 - Diagramme de composants	22
Figure 13 - Diagramme de déploiement	22
Figure 14 - Sélection des zones à corriger.....	24
Figure 15 - Diagramme de classe pour une session utilisateur	25
Figure 16 - Diagramme objet pour une session utilisateur.....	26
Figure 17 - Amélioration de la correction en ligne.....	27