

Kévin DESPOULAINS

Gaël GENDRON

Corentin GUILLOUX

Valentin FOUCHER

Enzo CRANCE

Charlotte RICHARD

Laure DU MESNILDOT

Timothée NEITTHOFFER

Elèves ingénieurs de l'INSA Rennes

Année universitaire 2018/2019

Rapport de conception

Projet 4INFO

Logiciel de génération de données d'apprentissage pour la reconnaissance d'écriture manuscrite

Encadrants

Bertrand COÜASNON (*IRISA*)

Erwan FOUCHÉ & Julien BOUVET (*Sopra Steria*)



Projet en collaboration avec

Jean-Yves LE CLERC

(*Archives départementales d'Ille-et-Vilaine*)

Sophie TARDIVEL (*Doptim*)



Table des matières

1	Introduction	1
2	Client	2
2.1	Choix du framework	2
2.2	Les différents composants de l'interface	2
2.3	Interactions	7
3	Serveur	9
3.1	Architecture générale	9
3.2	Traitement des données	11
3.3	Base de données	11
3.4	Interface avec le reconnaiseur	13
3.5	Contrôleur	14
4	Conclusion	15
5	Annexes	16

Chapitre 1

Introduction

Ce projet nous a été proposé par l'équipe [IntuiDoc](#) de l'[IRISA](#), en étroite collaboration avec la startup [Doptim](#) et avec le soutien de Jean-Yves LE CLERC, conservateur du patrimoine aux [archives départementales](#) d'Ille-et-Vilaine. Tout au long de l'année, nous serons encadrés par Bertrand COÜASNON, enseignant-chercheur membre d'IntuiDoc, Erwan FOUCHÉ, chef de projet chez [Sopra Steria](#) et Julien BOUVET, ingénieur chez Sopra Steria également. Nous serons aussi accompagnés par Sophie TARDIVEL, responsable et *data scientist* chez Doptim.

Ce rapport définira la structure de notre projet, ainsi que les moyens de conception mis en oeuvre pour répondre aux spécifications définies dans les précédents rapports. Dans ce rapport, nous rappellerons le contexte du projet qui justifie certains choix de notre conception. Nous avons décidé de conduire ce projet sur deux itérations. Nous aborderons donc pour chaque partie ce qui est intégré à la première version et à la seconde. Nous décrirons tout d'abord les choix de technologies pour la réalisation de la partie client du projet (IHM), son organisation, ainsi que ses interactions avec le serveur. Puis, nous parlerons de la structure du serveur, tout d'abord d'un point de vue général, ce serveur comportant beaucoup d'éléments plus difficiles à visualiser graphiquement, puis pour chaque module individuellement. Nous expliquerons également les interactions entre ces modules. Enfin, nous décrirons le fonctionnement du projet à l'aide de diagrammes de séquence avant de conclure ce rapport.

Chapitre 2

Client

2.1 Choix du framework

Pour réaliser l'interface Web de notre application, nous avons opté pour la technologie [Angular](#).

Celui-ci nous servira à gérer l'aspect dynamique de l'application web réalisée en HTML et CSS, comme la navigation d'une page de l'interface à une autre ou le passage de paramètres entre pages. Angular est un framework web développé en TypeScript et est l'un des plus utilisés pour le développement *front-end* des applications web. Les principaux avantages d'Angular sont qu'il est mis à jour très régulièrement par les développeurs de Google, qu'il dispose d'une très bonne documentation et que le développement en TypeScript est stable, rapide et facile.

Angular, HTML et CSS sont des technologies largement utilisées par les développeurs et relativement simples d'utilisation. En outre, nous avons déjà été amenés à les utiliser dans des projets précédents, nous partons donc avec des bases. Nous avons fait le choix de la simplicité et de l'utilisation de technologies que nous avons déjà abordées afin de faciliter et d'accélérer la réalisation de l'interface, pour être en mesure de proposer un prototype fonctionnel pour la première itération le 27 février.

2.2 Les différents composants de l'interface

Première itération Pour la première itération, l'interface est réduite à sa forme la plus simple qui répond au cahier des charges, à savoir ouvrir un document de travail et valider ou invalider les transcriptions. L'IHM aura donc deux composants : la page d'accueil et la page de validation des annotations.

La page d'accueil permet à l'utilisateur de choisir le projet sur lequel il veut travailler ainsi que d'en créer de nouveaux en choisissant les documents qui le composent. Un projet possède plusieurs documents qui possèdent eux-mêmes plusieurs pages contenant les imagettes (*PEA_GEN_6*). Cette page possède une liste des projets qui ont été ouverts auparavant dans l'application, ainsi qu'un bouton pour en créer un nouveau. Un clic sur ce bouton ouvre une nouvelle fenêtre pour que l'utilisateur parcoure ses documents et fasse son choix (spécifications *PEA_GEN_7* et *PEA_GEN_8*).

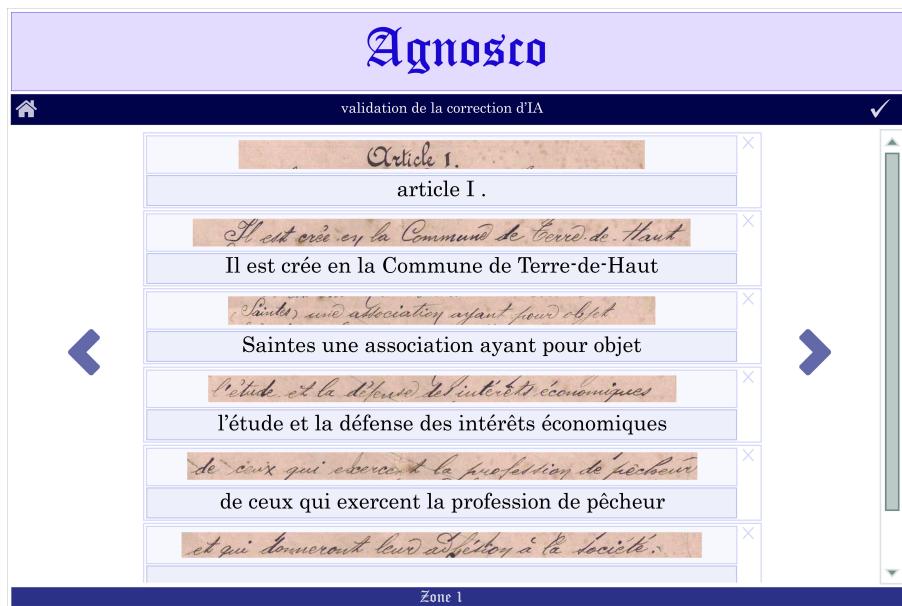
Figure 1 : Maquette de la page d'accueil de l'IHM



La page de validation (spécifications *PVAL_1* à *PVAL_7*), comme son nom l'indique, permet de valider ou d'invalider les transcriptions de la base de données relatives au document ouvert (*PEA_GEN_1*). Ces transcriptions ayant été réalisées par des humains, elles sont considérées comme vraies et l'utilisateur pourrait les invalider seulement s'il manque des mots ou si l'imagette est jugée peu pertinente pour l'apprentissage (par exemple, si le texte est presque illisible ou si l'imagette ne contient pas de texte du tout).

Pour ce faire, les imagettes sont affichées les unes à la suite des autres et chacune possède sa transcription située dans une zone de texte en dessous. Les imagettes sont affichées par groupe de cinq ou six. Une croix à côté de chaque imagette permet à l'utilisateur d'invalider le couple imagette-transcription correspondant. Un simple appui sur la touche Entrée valide l'ensemble des couples affichés sur la page, afin de permettre à l'utilisateur de réaliser la validation le plus rapidement possible. Des flèches gauche et droite permettent de se déplacer dans les groupes d'imagettes pour parcourir l'ensemble des pages du document. Enfin, une icône en haut à gauche renvoie à la page d'accueil.

Figure 2 : Maquette de la page de validation des annotations



Deuxième itération Pour la deuxième itération, nous implémenterons les fonctionnalités supplémentaires décrites dans les rapports précédents.

Nous proposerons tout d'abord une page de découpe des zones ou des paragraphes du document à l'aide d'outils graphiques regroupés dans une zone à droite de la page (spécifications *PDEC_OD_1* à *PDEC_OD_11*). Il y aura un outil pour créer une nouvelle sélection sous la forme d'un quadrilatère, un outil pour déplacer une sélection existante, un autre pour zoomer et dézoomer, un pour annuler ou refaire la dernière action, un outil pour réinitialiser la découpe et supprimer toutes les sélections réalisées et enfin, un dernier outil "premier jet" permettant de lancer un reconnaîtreur de lignes sur les zones découpées. Ce dernier permet de vérifier que les découpes ont bien été réalisées, soit que les lignes sont bien reconnues dans leur intégralité et qu'aucune partie du texte scanné n'est manquante.

La bannière en haut de la page contiendra plusieurs boutons de navigation. Un premier bouton permettra de revenir à l'accueil, comme dans toutes les pages de l'interface. Un autre exporterà la page découpée vers un reconnaîtreur d'écriture manuscrite en envoyant les couples imagettes - transcriptions au reconnaîtreur (*PDEC_OD_12*). Tout à droite, un autre bouton permettra à l'utilisateur de passer à la page de transcription manuscrite (*PDEC_OD_12*).

Figure 3 : Maquette de la page de découpe des zones



Lors de l'ouverture d'un nouveau document de travail, il devra être possible d'utiliser des manuscrits scannés qui ne possèdent pas de vérités-terrain. L'utilisateur aura alors la possibilité d'annoter le manuscrit manuellement ou de le faire passer par un reconnaîtreur d'écriture après avoir découpé les zones du manuscrit via la page de découpe de l'interface.

L'interface possèdera donc une page d'annotation manuelle (*PEA_GEN_2*). Les imagettes d'une même zone seront affichées à la suite dans la fenêtre centrale, avec une zone de texte pour la transcription située sous chaque imagette. Le curseur sera positionné automatiquement sur la première transcription affichée (*PEMA_1*), l'utilisateur devra taper la transcription manuellement au clavier puis appuyer sur la touche Entrée pour passer à la transcription suivante (*PEMA_2*). Lorsque toutes les imagettes d'une zone ont été annotées, on passe automatiquement aux imagettes de la zone suivante à l'appui sur Entrée de la dernière transcription. Le but est de minimiser les actions requises par l'utilisateur, pour qu'il n'utilise que son clavier. Il peut néanmoins positionner son curseur sur la transcription de son choix à l'aide de sa souris.

Une croix à droite de chaque imagette permet de supprimer le couple imagette-transcription de la base d'apprentissage si le couple est jugé peu pertinent (*PEMA_6*). Après l'appui sur ce bouton, le couple est flouté mais toujours visible et la croix est remplacée par une flèche montante qui permet de réintroduire le couple supprimé dans la base d'apprentissage.

Cette page de l'interface possèdera également une fenêtre à droite montrant la page courante en entier, avec toutes ses zones de découpe affichées en surbrillance. Un rectangle parcourera la page au fur et à mesure de la transcription pour indiquer la position courante dans la page.

Sous cette fenêtre de visualisation de la page, l'interface proposera une fenêtre de navigation dans les différents documents, permettant à l'utilisateur d'ouvrir un autre document de travail à tout moment.

Dans la bannière en haut de la page, un bouton permettra à l'utilisateur de revenir à la page de découpe des zones ou encore d'avancer à la page de validation finale des transcriptions, tandis qu'un autre renverra à l'accueil.

Figure 4 : Maquette de la page d'annotation manuelle



Si l'utilisateur souhaite faire passer le manuscrit par le reconnaiseur plutôt que de l'annoter manuellement, il pourra visualiser et corriger les résultats de la reconnaissance via la page dédiée de l'interface (*PEA_GEN_3*). Cette page sera sensiblement similaire à la page d'annotation manuelle, la différence principale résidant dans le fait que des transcriptions réalisées par le reconnaiseur seront déjà présentes sous chaque imagette. Le curseur se positionnera automatiquement sur la première transcription et un simple appui sur la touche Entrée passera à la transcription suivante. L'utilisateur pourra modifier manuellement la transcription courante en tapant ses modifications au clavier (spécifications *PCORIA_1* à *PCORIA_5*).

Dans la bannière du haut de la page, un bouton renverra à l'accueil et un autre permettra la navigation vers la page de validation finale des transcriptions.

Figure 5 : Maquette de la page de visualisation des transcriptions du reconnaiseur

La maquette de la page de visualisation des transcriptions du reconnaiseur est une interface web nommée "Agnosco". Le titre "Agnosco" est en haut à gauche. En haut à droite, il y a un bouton "correction de l'apprentissage / validation". La page est divisée en deux sections principales :

- Section principale (gauche) :** Affiche une liste de zones de transcription avec leur contenu original et leur transcription automatique. Les zones sont numérotées de 1 à 5. Par exemple :
 - Zone 1 : Article 1. article I .
 - Zone 2 : Il est crée en la Commune de Terre-de-Haut
 - Zone 3 : Saintes une association ayant pour objet
 - Zone 4 : l'étude et la défense des intérêts économiques
 - Zone 5 : de ceux qui exercent la profession de pêcheur
- Section secondaire (droite) :** Affiche "Document 2.1" avec une miniature du document original et une liste de zones annotées par un rectangle bleu. En dessous, il y a un bouton "Page1" et une section "Documents du même projet" avec une liste de documents.

En bas de l'écran, il y a un bouton "Zone 1" et une barre de navigation avec les nombres 1, 2, 3, 4, 5.

Enfin, la dernière page de l'interface sera la page de validation qui a déjà été réalisée lors de la première itération. Deux modifications y seront apportées pour la deuxième itération.

Tout d'abord, une fenêtre sera ajoutée pour visualiser la page courante en miniature avec ses zones de découpe affichées par dessus ainsi qu'un rectangle montrant la position courante de l'utilisateur, afin que ce dernier ait constamment accès à sa position dans le document.

Ensuite, la bannière en haut de la page affichera si les transcriptions ont été réalisées manuellement par un humain ou par le reconnaiseur d'écriture, car l'attention fournie par l'utilisateur devrait être moindre pour un manuscrit annoté manuellement où les fautes sont théoriquement moins nombreuses. Savoir si la transcription a été faite manuellement ou par le reconnaiseur permettrait d'augmenter la rapidité de la validation finale.

Figure 6 : Maquette de la page de validation finale lors de la deuxième itération

La maquette de la page de validation finale lors de la deuxième itération est une interface web nommée "Agnosco". Le titre "Agnosco" est en haut à gauche. En haut à droite, il y a un bouton "validation de la correction d'IA" et un bouton avec une coche. La page est divisée en deux sections principales :

- Section principale (gauche) :** Affiche une liste de zones de transcription avec leur contenu original et leur transcription automatique. Les zones sont numérotées de 1 à 5. Par exemple :
 - Zone 1 : Article 1. article I .
 - Zone 2 : Il est crée en la Commune de Terre-de-Haut
 - Zone 3 : Saintes une association ayant pour objet
 - Zone 4 : l'étude et la défense des intérêts économiques
 - Zone 5 : de ceux qui exercent la profession de pêcheur
- Section secondaire (droite) :** Affiche une miniature de la page de validation avec un curseur bleu indiquant la position de l'utilisateur. En dessous, il y a un bouton avec une flèche vers la droite.

En bas de l'écran, il y a un bouton "Zone 1".

En résumé, à la fin de la deuxième itération, nous aurons cinq composants différents : la page d'accueil, la page de découpe des zones du document, la page d'annotation manuelle, la page de visualisation de la reconnaissance et la page de validation finale des transcriptions.

2.3 Interactions

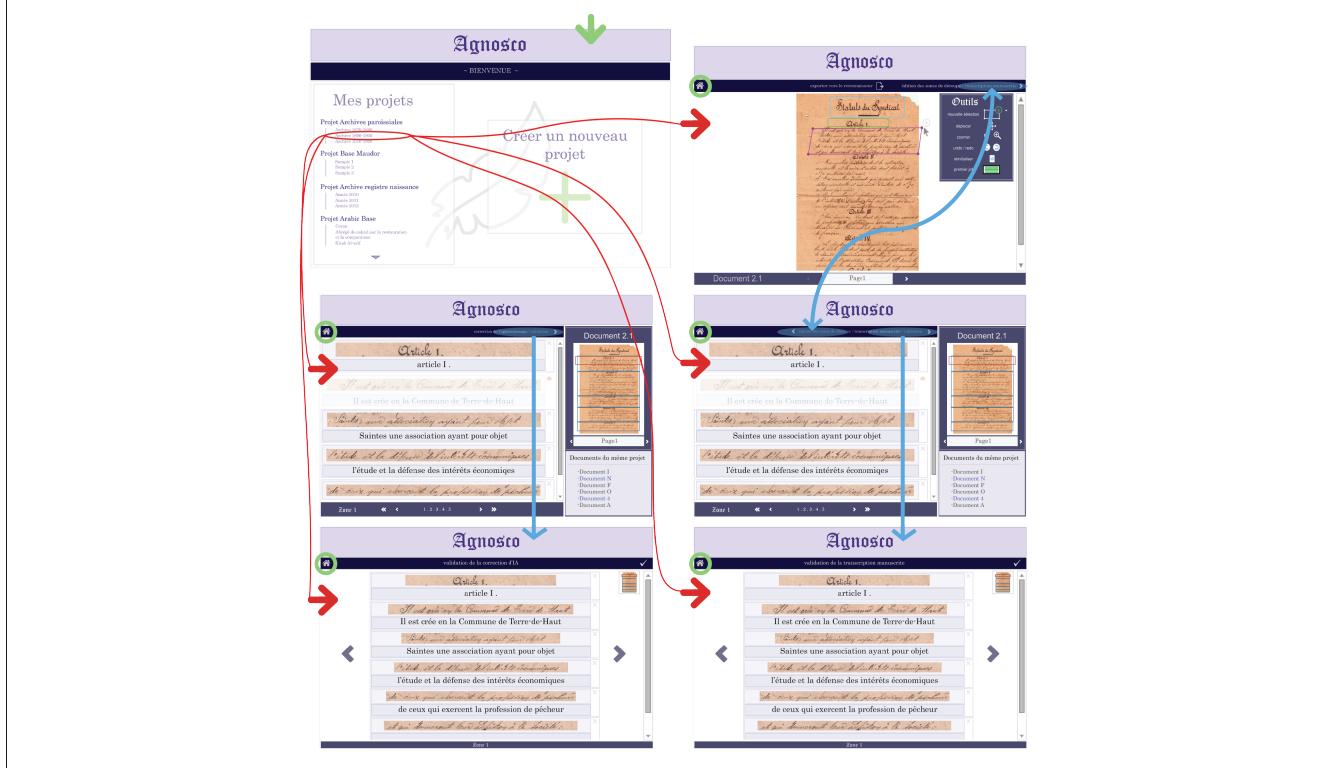
Interactions entre les composants de l'interface L'utilisateur peut naviguer entre les différents composants via les boutons de l'interface.

Tout d'abord, une icône permettant de revenir à la page d'accueil sera présente en haut à gauche de chaque page - à l'exception de la page d'accueil. Sur cette dernière, le choix du document ouvre celui-ci dans la page de l'interface qui lui correspond, soit :

- la page de découpe si aucun travail n'a été réalisé au préalable sur ce document ;
- la page d'annotation manuelle si les découpes ont déjà été réalisées et que l'utilisateur n'a pas exporté le document vers le reconnaiseur ;
- la page de visualisation des transcriptions du reconnaiseur si une reconnaissance a été réalisée sur le document ;
- ou la page de validation finale si toutes les transcriptions ont déjà été renseignées.

Des boutons de navigation dans les différentes pages sont également présents sur certaines pages, comme décrit dans la partie précédente. Les interactions sont visibles dans le schéma ci-dessous :

Figure 7 : Schéma des interactions entre les différents composants de l'IHM



Interactions entre le front-end et le back-end L'interface affiche les informations de la base de données (les couples imagettes - transcriptions) et les modifie. Pour ce faire, elle est reliée au connecteur central via sa ressource Rest. Le connecteur traite les demandes de l'IHM et les envoie au connecteur de la base de données. Celui-ci effectue les ordres de l'utilisateur sur la base et renvoie les imagettes et les transcriptions au connecteur central qui les fait suivre à l'IHM. Les modifications (ajout ou modification d'une transcription ou suppression d'un couple) sont enregistrées localement puis envoyées à la base de données lors du changement de page et du chargement d'une nouvelle page. Nous n'avons donc pas de bouton spécifique pour enregistrer les modifications.

API Rest Pour communiquer avec le *back-end*, nous utiliserons une API Rest dont les requêtes sont listées ci-dessous.

Général

GET /base/projectsAndDocuments

Renvoie la liste des noms des projets, contenant pour chaque projet une liste des noms des documents qui le composent.

DELETE /base/deleteDocument/{name}

Supprime le document de la base qui porte le nom donné, ainsi que son contenu.

GET /base/availableRecognisers

Renvoie la liste des noms des reconnaiseurs disponibles sur le serveur.

POST /base/exportRecogniserExamples/{name}

Récupère tous les exemples de la base qui sont contenus dans des projets utilisant le reconnaiseur dont le nom est passé en paramètre. Les exemples sont triés, ne sont retenus que ceux qui sont définis comme utilisables et validés. Ensuite, ces exemples sont exportés en tant que lot d'entraînement, vers le format d'entrée associé au reconnaiseur en question.

Annotation / Validation

GET /base/documentPages/{name}

Renvoie la liste des identifiants en base de données des pages qui composent un document dont le nom est donné en paramètre.

GET /base/pageData/{id}

Renvoie l'image associée à la page dont l'identifiant est donné en paramètre, ainsi que la liste des imagettes et des transcriptions des exemples qui la composent.

POST /base/saveExampleEdits/

Enregistre dans la base de données les modifications de transcriptions décrites par l'objet JSON associé à la requête.

PUT /base/disableExample/{id}

Rend l'exemple dont l'identifiant est donné en paramètre inutilisable, i.e. l'utilisateur juge que l'image est trop parasitée ou que l'exemple n'est pas pertinent.

PUT /base/enableExample/{id}

Rend l'exemple dont l'identifiant est donné en paramètre utilisable (principalement utilisé pour annuler l'action décrite ci-dessus).

POST /base/validateExamples/

Valide tous les exemples dont les identifiants sont fournis dans une liste JSON associée à la requête.

Découpe

GET /base/documentPagesWithImages/{name}

Renvoie la liste des identifiants et des images associés aux pages qui composent le document portant le nom donné.

POST /base/addDocumentGroundtruth/{name}

Ajoute à la base la vérité terrain donnée dans l'objet PiFF (donc JSON) associé à la requête, et l'associe au document dont le nom est donné en paramètre.

GET /base/recogniseImages/{name}

Envoie la liste des imagettes contenues dans le document donné au reconnaiseur associé au projet contenant ce document. La réponse à cette requête est une liste de transcriptions trouvées par le reconnaiseur.

Chapitre 3

Serveur

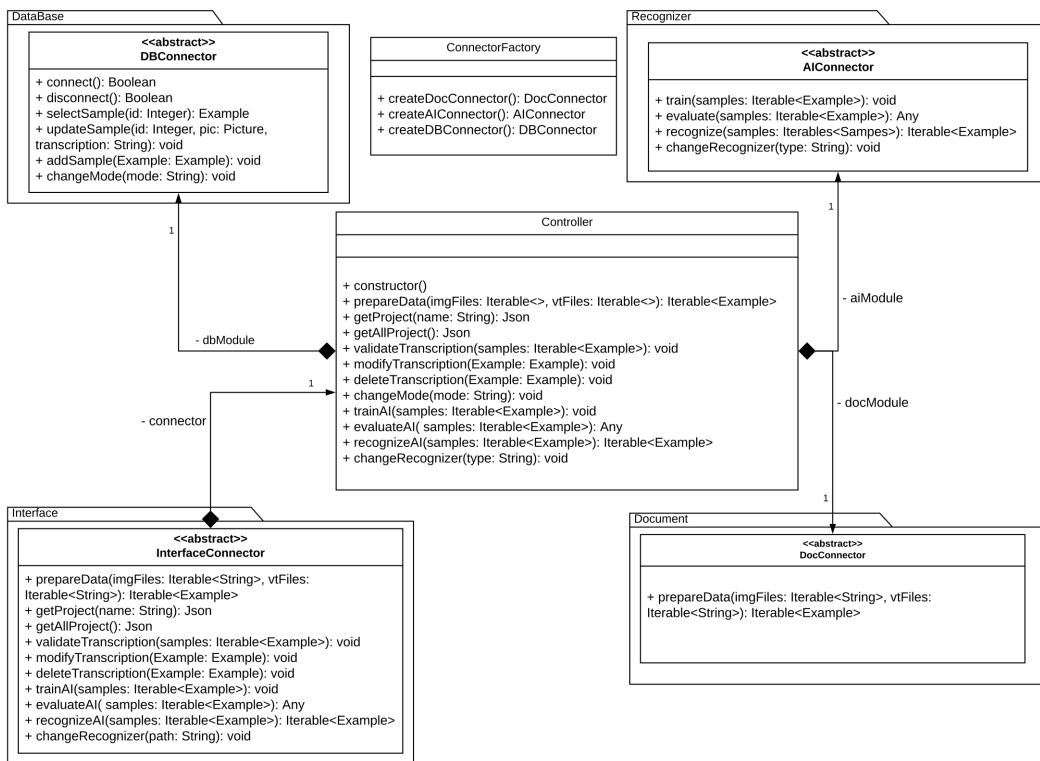
Ce projet a pour but de fournir un programme permettant de concevoir des bases d'apprentissage automatiquement pour l'entraînement de systèmes de reconnaissance d'écriture manuscrite. Il sera notamment exploité par les [archives d'Illes-et-Vilaine](#) ainsi que la startup [Doptim](#). Les reconnaiseurs utilisés pouvant être multiples, il faut que ce projet puisse facilement évoluer, qu'une partie du projet puisse être remplacé par un morceau plus adapté au reconnaisseur choisi. Ainsi, tous les modules de notre projet et non seulement l'interface avec le reconnaiseur doivent pouvoir être remplacés par l'implémentation choisie par l'utilisateur. Par exemple, nous avons choisi une base de données intégrée avec *sqlite* mais celle-ci ne peut gérer facilement l'accès concurrentiel à la base de données, ou gérer efficacement une grande quantité de données. Ainsi, l'utilisateur pourrait choisir d'utiliser une autre base de données comme *MySQL* ou *MongoDB*. Nous avons donc du prendre en compte dans l'architecture l'aspect interchangeable de nos modules.

3.1 Architecture générale

Le serveur de ce projet est composé de trois principaux modules représentant les différents besoins du projet. Ainsi, il nous faut traiter les données d'entrées fournies par l'utilisateur sous la forme d'un document scanné et possiblement d'une vérité terrain afin de les transformer en données utilisables par les reconnaiseurs. Il nous faut également pouvoir stocker les bases d'apprentissage qui constituent le coeur de notre projet. Enfin, ces bases ne serviraient à rien s'il n'était pas possible d'interfacer notre projet avec le reconnaiseur de l'utilisateur.

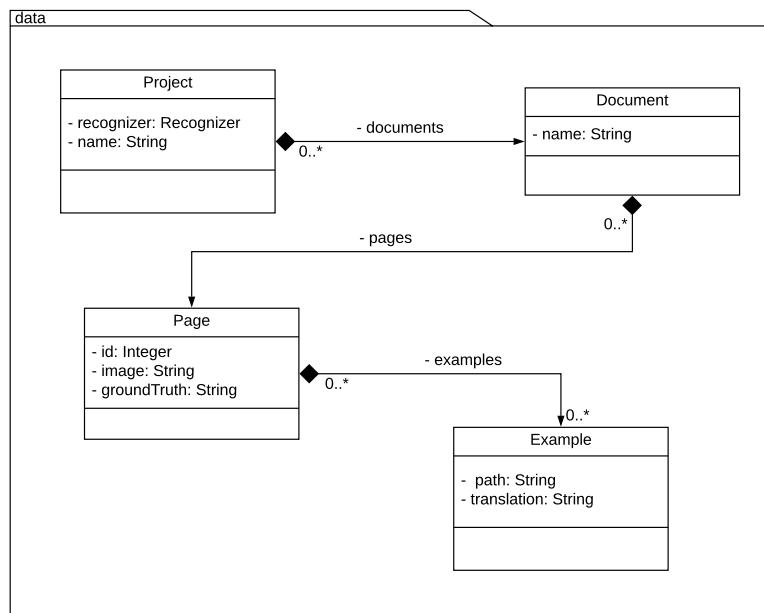
Notre projet étant composé de parties bien distinctes, la mise en place de modules indépendants et pouvant être remplacés par l'utilisateur n'a donc pas posé de problèmes. Nous avons donc créé une structure constituée des différents packages correspondant aux fonctionnalités ainsi qu'une interface faisant le lien entre tous. De cette manière, chaque partie est détachée de l'ensemble global et l'interface centrale qu'on appellera **Controller** fera appel aux méthodes nécessaires des différents packages afin de répondre aux demandes de l'utilisateur. Les packages auront alors une interface à implémenter permettant une utilisation indépendante de l'implémentation.

Architecture des modules avec le connecteur



Dans notre projet, nous aurons également besoin de représenter les objets avec lesquels nous travaillons. Ainsi, nous avons choisi d'implémenter des classes de données pour représenter les exemples d'apprentissage (Example), les pages des documents utilisés (Page), les-dits documents (Document) et enfin les projets (Project) car on peut imaginer que l'utilisateur puisse vouloir avoir un projet sur des archives paroissiales et un autre sur des textes arabes anciens.

Structure du package de données



L'attribut Recogniser est converti en String dans la base de données en passant par un enum recensant les différents reconnaiseurs existant. Ces classes de données sont liées à la structure de la base de données et seront donc expliquées plus profondément dans cette partie.

3.2 Traitement des données

Pour ce qui est du traitement des données, il nous faut un *package* pour chacune des deux tâches concernées, à savoir la lecture des fichiers d'entrée ainsi que la découpe d'image.

3.2.1 Lecture des fichiers d'entrée : *package* input

Comme expliqué dans le dernier rapport, nous avons choisi de ne traiter qu'un seul format dans notre logiciel, le format PiFF. Pour pouvoir lire les fichiers d'entrée, on doit construire une représentation des données contenues dans ces derniers sous la forme d'objets. Ceci constituera le *package* piff. Il définit une classe PiFF, qui contient des pages (PiFFPage), qui elles-mêmes contiennent des portions de texte (PiFFEelement). Ces classes peuvent être converties au format JSON (bibliothèque org.json), ce qui permet d'exporter les objets vers un fichier PiFF afin de répondre à la spécification PR_FO_1 pour l'utilisation du format PiFF en interne.

L'utilisateur doit toutefois pouvoir utiliser d'autres formats afin de permettre l'évolution de notre logiciel comme spécifié dans les précédents rapports (GEN_EVO). Pour cette raison, nous avons un *package* converters contenant une interface, PiFFConverter, qui permet de convertir un fichier quelconque en objet PiFF. Nous en fournissons une implémentation, l'objet GEDIToPiFFConverter, qui comme précisé dans les précédents rapports (PR_FO_2), permet au logiciel de lire le format GEDI. Celui-ci est utilisé notamment par la base de données Maurdor, présentée dans les précédents rapports, et à laquelle nous avons accès pour nos tests. L'utilisateur peut rajouter autant d'implémentations qu'il le souhaite.

En plus de ces deux *packages*, nous proposons un objet PiFFReader (singleton), qui permettra d'ouvrir un fichier, et d'utiliser les concepts présentés ci-dessus. Plus précisément, il permettra de lister des implémentations de PiFFConverter, et de les appeler une par une sur le fichier d'entrée pour réussir à le lire.

3.2.2 Découpe des images : *package* processing

Pour la découpe d'image, il nous faut une classe ImageProcessing, qui appelle les méthodes de la bibliothèque OpenCV, que nous avons choisie précédemment. Cette bibliothèque nous permet de découper les images afin d'obtenir les imagettes selon les lignes ou les paragraphes (PR_TR_2 et PR_TR_4). Après la découpe, les imagettes sont associées au texte pour former des exemples, que nous avons modélisés par une classe Example (spécifications PR_RE_1 et PR_RE_2).

La phase de découpe étant automatique, il nous faut faire appel à un détecteur de lignes. C'est la fonction du *package* linedetection. Nous y plaçons une interface LineDetector, qui permet de trouver les lignes de texte dans un objet PiFF (spécification PR_TR_1). Le détecteur de lignes utilisé pour le projet est fourni par l'encadrant, et fonctionne sous Linux. Les membres du groupe peuvent pourtant travailler sous Windows ou macOS. Pour faciliter le développement du logiciel, nous avons donc choisi de fournir une implémentation, la classe BlurLineDetector (nom dû à la méthode de détection), basée sur des connexions réseau, afin de pouvoir faire fonctionner le serveur sous n'importe quel système d'exploitation, avec un petit module serveur sous Linux qui appelle simplement l'exécutable. Ce petit module ne fait pas partie du cahier des charges mais sera développé pour les tests. L'utilisateur, ici aussi, pourra rajouter ses propres méthodes de détection de lignes s'il le souhaite.

Version

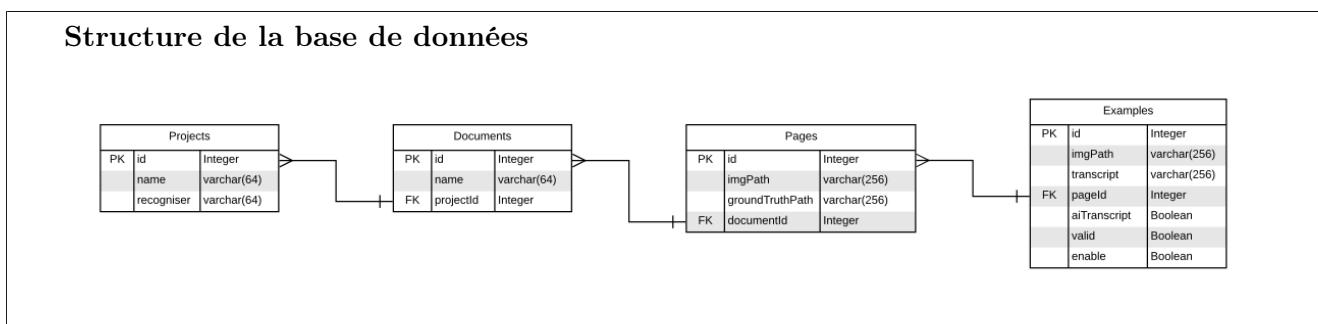
3.3 Base de données

Le Système de Gestion de Bases de Données (SGBD) choisi est SQLite. La bibliothèque Java SQLite, une implémentation de l'API JDBC (Java DataBase Connectivity), nous servira à appeler des commandes SQL directement depuis notre programme. Il nous faut cependant remanier les relations entre les tables de la base de données car en nous concertant avec l'équipe sur l'ergonomie à apporter ainsi que la forme que

devaient prendre les données, nous avons remarqué que notre précédente organisation ne pouvait fournir tous les éléments nécessaires. En effet, dans les précédents rapports, la forme que devait prendre l'IHM quand aux données venant du server et dont elle aurait besoin était plutôt floue. C'est en se penchant plus en avant lors de ce rapport que nous l'avons donc remarqué.

3.3.1 Architecture

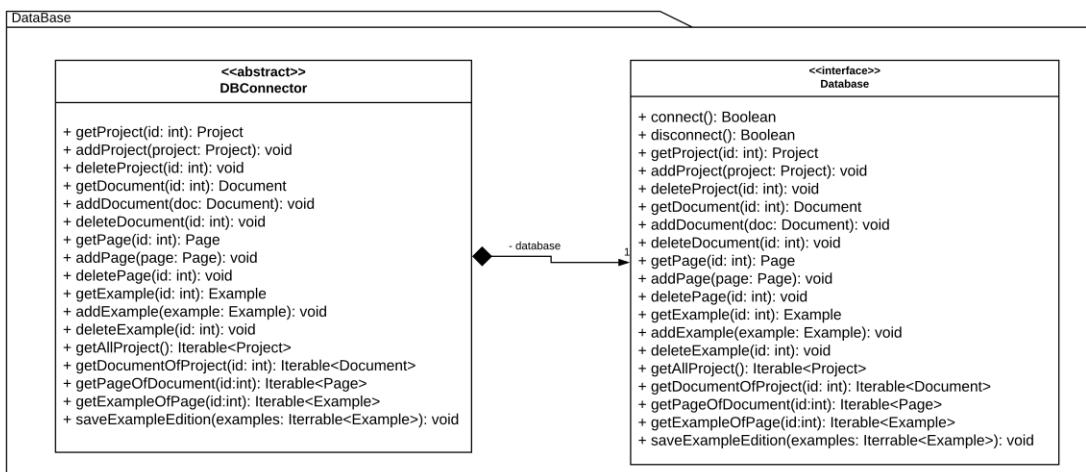
La nouvelle base de données se base sur les classes de données présentées dans la première partie. Elles sont donc composées d'une table `Projects` qui contient les projets existants ainsi que le `Recogniser` associé. Celui-ci est représenté sous la forme d'une `String` car un `enum` sera mis en place afin de constituer les différents reconnaiseurs possibles et qu'on peut restorer l'`enum` facilement à partir d'un `String`. Une autre table `Documents` représente les différents documents qui constituent notre base, par exemple, "Archives paroissiales de 1870-1872". Ceux-ci sont donc représentés par un nom et sont liés à un `Project`. Une troisième table nous permet de conserver les `Pages` des `Documents`. Elles sont donc liées à une image (le scan de la page) et à une vérité terrain (la description de l'image) qui sont stockées au travers de leur chemin dans le système de fichier pour les raisons expliquées dans les rapports précédents. Les `Pages` sont également associées au `Document` dont elles sont issues. Enfin, une dernière table `Examples` stocke les `exemples` générés par la découpe. Ils sont constitués du chemin vers l'imagette correspondante ainsi que la transcription de celle-ci sachant que cette transcription peut être nulle dans le cas où on souhaite utiliser le `reconnaisseur` pour proposer une transcription ou quand la vérité terrain n'a pas été fournie et qu'il faut la construire. Les `Examples` sont également liés à la `Page` d'où ils ont été découpés.



Cette structuration nous permet de hiérarchiser les exemples d'apprentissage générés par notre logiciel et donc de pouvoir fournir une plus grande ergonomie au sein de l'application client.

Le module de la base de données est constitué d'une interface représentant cette base de données ainsi que les opérations qui seront effectuées dessus. Ainsi, il est possible de récupérer les différents instances de la hiérarchie de donnée. Nous avons préféré garder les requêtes sur la base de données simples et élémentaires afin de permettre plus de flexibilité sur les opérations qui les utiliserons ensuite. De cette manière, plusieurs fonctions du Controller pourront appeler la même fonction dans la base plutôt que d'avoir des demandes précises sur la base de données, mais d'avoir à écrire des requêtes particulières pour chaque demande. De cette manière, l'évolution du projet est simplifiée en permettant l'établissement simple de nouvelles fonctions dans le Controller sans avoir à rajouter de requêtes dans la base de données.

Architecture de la base de données

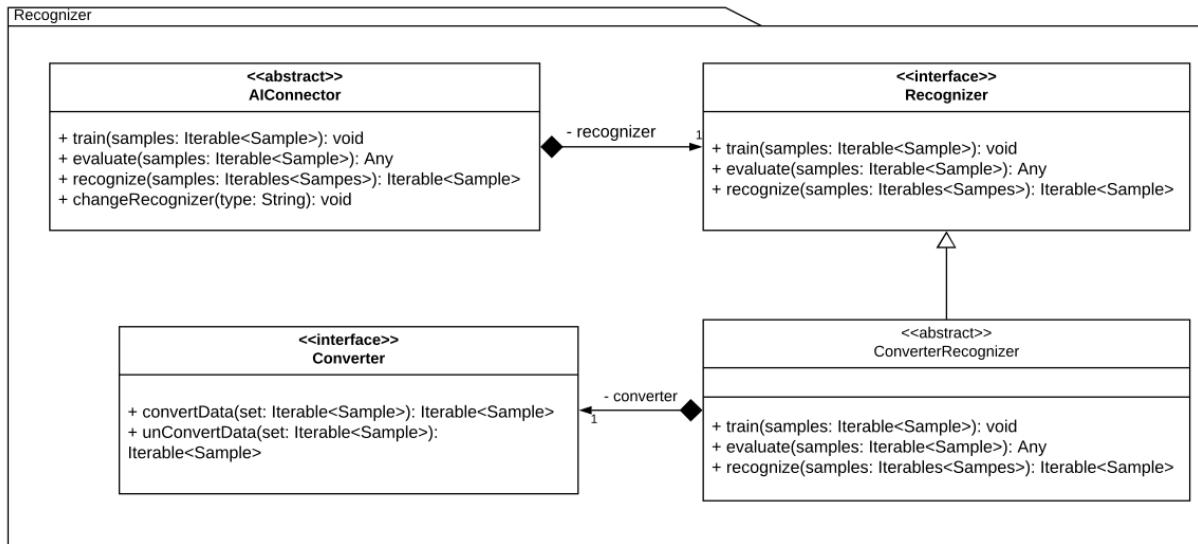


Version Comme l'ensemble des méthodes de la base de données est nécessaire pour les appels de l'API Rest, toute la classe devra être implémentée. Pour la seconde itération de notre projet, de nouvelles méthodes ne seront pas nécessaires comme nous réutiliserons celles déjà créées pour les ajouts de méthodes dans le contrôleur.

3.4 Interface avec le reconnaiseur

Cette partie du projet a pour objectif de lier la base d'apprentissage de notre logiciel avec le reconnaiseur choisi par l'utilisateur. Ainsi, cette partie étant très liée à l'utilisateur, il faut lui permettre de pouvoir facilement attacher le reconnaiseur de son choix au logiciel. C'est avec cette contrainte en tête que nous avons conçu l'architecture de cette partie.

Architecture de l'interface avec le reconnaiseur



Architecture Cette partie du projet contient comme toutes les autres un connecteur qui la lie au Connector principal. Ce connecteur possède un Reconnaître afin de pouvoir effectuer les opérations classiques d'entraînement, d'évaluation de celui-ci ainsi que lui demander d'effectuer une transcription. Il permet également de changer de reconnaiseur quand l'utilisateur souhaite utiliser un autre type de reconnaiseur.

Le reconnaiseur est représenté par une interface `Recogniser` qui permet trois actions possibles : entraîner le reconnaiseur à partir d'un ensemble d'apprentissage, évaluer ce même reconnaiseur sur un ensemble de validation, et enfin, lui demander de transcrire un ensemble non étiqueté. Ces fonctions ont été construites afin de répondre aux spécifications `IR_AP`, `IR_EV` ainsi que `IR_TR`. Cette interface permet notamment à l'utilisateur, de connecter son propre reconnaiseur, et même s'il le souhaite, d'en créer un au sein de l'application. Elle est également la seule qu'il est obligé d'implémenter pour faire fonctionner l'ensemble. Cependant, il faut dans certains cas convertir les données dans la base de donnée en données compréhensibles par le reconnaiseur. Ainsi, deux autres interfaces sont nécessaires : une interface `Converter` permettant la conversion des données entrantes et sortantes du reconnaiseur (`IR_CV`) ainsi qu'une classe abstraite `ConverterRecogniser` héritant de `Recogniser` et possédant un convertisseur adapté au reconnaiseur utilisé. En effet, le reconnaiseur peut avoir besoin d'un formatage des images en entrée, mais également de la transcription associée. Par exemple, nous implémenterons la structure décrite précédemment pour un reconnaiseur particulier : [Laia](#). Ce reconnaiseur demande de transformer les images afin qu'elles aient la même hauteur en pixel et que la transcription soit traduite dans les symboles qu'il peut reconnaître et organisée d'un certaine façon. Par exemple, il faut remplacer les ' ' par ' <space>' et que l'identifiant de l'image correspondante à la transcription soit indiqué avant celle-ci dans un fichier. Le convertisseur va donc se charger de mettre en forme les données et de créer ce qui est nécessaire.

Version Dans la première version délivrée le 27 Février, la possibilité d'entraîner le reconnaiseur doit être implémentée ainsi que celle de changer de reconnaiseur. Dans un second temps, il sera possible de faire transcrire les imagettes par le reconnaiseur et de faire l'évaluation de celui-ci. Les données remontées lors de l'évaluation sont laissées libres à l'utilisateur puis la fonction rend un objet non défini à l'avance qui sera sous la forme d'un `JSON` et qui contiendra les informations choisies par l'utilisateur dans son implémentation. Il pourra alors obtenir le nombre et la fréquence des erreurs, sur quelles lettres celles-ci surviennent et ainsi de suite.

3.5 Contrôleur

3.5.1 Architecture

Le rôle du contrôleur est de mettre en relation les 4 parties du projet : le *client*, i.e. l'interface homme-machine, la base de données, ainsi que le traitement des données. Pour ce faire, nous avons 4 classes abstraites, un connecteur par *package* du projet, chacun possédant des méthodes générales, qui appellent d'autres méthodes de leurs *packages* respectifs. Cela permet d'avoir un objet `Controller` central qui manipule tous les éléments du projet avec un code très lisible. Nous fournissons une implémentation de chaque connecteur, pour le bon fonctionnement du projet. Une fabrique est également prévue, permettant de choisir l'implémentation des connecteurs de chaque partie sans avoir à modifier le code du connecteur central. En effet, le contrôleur récupèrera les instances des connecteurs à partir de cette fabrique. Cela permet à l'utilisateur de changer d'instance de connecteur en changeant une ligne de code seulement.

Pour la première version de notre projet, le `Controller` devra permettre de relier les différents modules au travers de leurs interfaces de connection ainsi que permettre les exécutions des demandes en provenance de l'utilisateur, c'est à dire, le client.

3.5.2 Interactions

Le connecteur central reçoit donc, au travers du connecteur en relation avec le client les demandes de l'utilisateur et, afin d'exécuter ces demandes, appelle au travers des connecteurs des autres modules, les méthodes nécessaires. Le `Controller` sert alors d'intermédiaire entre l'utilisateur et les différents modules. Il permet notamment cette structure modulaire en détachant les appels de méthode des classes réelles qui peuvent alors être changées selon le bon vouloir de l'utilisateur.

Chapitre 4

Conclusion

Dans le cadre de notre projet de 4e année à l'INSA, nous avons pour objectif de conduire un projet depuis la rédaction du cahier des charges jusqu'à sa livraison finale en passant par les spécifications, la conception et le développement. Une fois le cahier des charges et les spécifications réalisées, nous avons rédigé le présent rapport, détaillant la conception de notre projet.

Nous avons découpé ce rapport en deux parties principales : le côté client et le côté serveur, après avoir rappelé le contexte de notre projet de génération de bases d'apprentissage pour la reconnaissance d'écriture.

Dans la première partie, nous avons expliqué nos choix des technologies utilisées pour la réalisation de l'interface. Nous avons ensuite détaillé les différents composants de l'IHM pour la première itération puis pour la deuxième, avant de décrire les interactions entre les pages de l'IHM et entre le *front-end* et le *back-end*.

La deuxième partie était axée sur le côté serveur de l'application. Nous avons tout d'abord décrit l'architecture générale de notre projet. Nous avons ensuite détaillé la conception des blocs de la découpe des images, de la base de données et de l'interface avec le reconnaiseur, en précisant dans chaque partie l'architecture correspondant aux deux itérations successives ainsi que les interactions de chaque bloc avec le reste de l'application.

Ce rapport est émaillé de schémas d'architecture, de diagrammes de classes ainsi que de maquettes de l'interface pour la clarté et la lisibilité, afin de permettre une meilleure compréhension à la lecture.

Le projet se poursuivra par une phase de développement afin de produire un premier rendu fonctionnel pour le 27 février.

Chapitre 5

Annexes

5.0.1 Codes de specification

Bloc 1 : Préparation des données	
Spécification	Description
PR_FO_1	Traiter le format PiFF en interne dans le logiciel
PR_FO_2	Fournir un convertisseur du format GEDI vers PiFF
PR_TR_1	Intégrer une fonction de détection de lignes au logiciel
PR_TR_2	Permettre un découpage des images en lignes
PR_TR_3	Localiser les paragraphes
PR_TR_4	Permettre un découpage des images en paragraphes
PR_RE_1	Associer les images à leur transcription
PR_RE_2	Associer la vérité terrain à une transcription
PR_RE_3	Permettre de générer une vérité terrain si besoin, grâce à un reconnaiseur

Bloc 2 : Stockage des données	
Spécification	Description
STO_VER	Stocker des imagettes associées à une vérité terrain
STO_USR	Stocker des imagettes associées à une transcription générée par l'utilisateur
STO_REC	Stocker des imagettes associées à une transcription générée par un reconnaiseur
STO_SEL	Fournir des méthodes pour accéder aux données stockées
STO_UPD	Fournir des méthodes pour modifier les données stockées
STO_INS	Fournir des méthodes pour pouvoir insérer des données à stocker
STO_DEL	Fournir des méthodes pour pouvoir supprimer des données stockées

Bloc 3 : Interface avec le reconnaiseur	
Spécification	Description
IR_CV	Convertir les données au format d'entrée du reconnaiseur
IR_AP	Fournir les données au reconnaiseur
IR_EV	Pouvoir lancer une évaluation du reconnaiseur
IR_TR	Pouvoir lancer une transcription d'un document par le reconnaiseur

Bloc 4 : Interface avec l'utilisateur	
Spécification	Description
PEA_GEN_1	Valider un ensemble d'annotations
PEA_GEN_2	Éditer manuellement les transcriptions
PEA_GEN_3	Corriger les annotations proposées par un reconnaiseur externe à l'application
PEA_GEN_4	Envoyer les modifications à la base de données lorsque la vérité-terrain d'une imagette est modifiée
PEA_GEN_5	Ignorer un couple imagette-transcription s'il n'est pas pertinent
PEA_GEN_6	Regrouper les documents en projets
PEA_GEN_7	Sélectionner d'abord le projet puis le document sur lequel l'utilisateur veut travailler à l'ouverture de l'application
PEA_GEN_8	Créer un nouveau projet
PEA_GEN_9	Basculer vers la page de découpe des zones
PEA_GEN_10	Basculer vers la page d'édition des annotations
PEA_GEN_11	Basculer vers la page de validation des transcriptions
PDEC_OD_1	Créer une nouvelle zone à l'aide d'un rectangle (outil "nouvelle sélection")
PDEC_OD_2	Pouvoir modifier la position des sommets des rectangles
PDEC_OD_3	Rajouter des sommets à la zone
PDEC_OD_4	Changer le type de la zone avec un menu déroulant
PDEC_OD_5	Déplacer la zone sélectionnée sur le document (outil "déplacer")
PDEC_OD_6	Zoomer et dézoomer sur le document (outils "zoom +" et "zoom -")
PDEC_OD_7	Annuler la dernière action (outil "annuler")
PDEC_OD_8	Refaire l'action annulée précédemment (outil "refaire")
PDEC_OD_9	Supprime toutes les zones de la page pour retourner au document vierge (outil "réinitialiser")
PDEC_OD_10	Applique un détecteur de lignes sur la zone sélectionnée (outil "appliquer la détection de lignes sur la zone")
PDEC_OD_11	Continuer la découpe du document sur la page suivante
PDEC_OD_12	Passer à l'édition des annotations sur la page qu'il vient de découper
PDEC_OD_13	Exporter la page découpée au format PiFF afin de soumettre les données à un reconnaiseur externe à l'application
PDEC_OD_14	Posséder un bouton de retour au menu principal
PDEC_OD_15	Permettre à l'utilisateur de se déplacer sur le manuscrit à l'aide d'un scroll horizontal et vertical
PEMA_1	Placer le curseur sur la première imagette ne possédant pas de transcription
PEMA_2	Positionner le curseur sur l'annotation suivante en appuyant sur Entrée
PEMA_3	Proposer un raccourci clavier permettant de basculer vers la prochaine imagette sans vérité-terrain
PEMA_4	Posséder un bouton intitulé "modifier les zones du manuscrit"
PEMA_5	Afficher la liste des imagettes du document découpé
PEMA_6	Ignorer un couple imagette-transcription s'il n'est pas pertinent
PEMA_7	Basculer vers la page de validation des transcriptions
PCORIA_1	Valider les transcriptions zone par zone et passer à la zone suivante avec un simple appui sur Entrée
PCORIA_2	Pouvoir modifier une annotation fausse en cliquant dessus pour y positionner son curseur et en effectuant ses modifications manuellement
PCORIA_3	La zone de visualisation des imagettes se présente de la même manière que sur la page d'édition manuelle des transcriptions
PCORIA_4	Posséder également la fonctionnalité de mise à l'écart d'un couple imagette-transcription
PCORIA_5	Basculer vers la page de validation des transcriptions

Spécification	Description
PVAL_1	Accéder à la page de validation depuis le menu principal
PVAL_2	Accéder à cette page de validation depuis les pages d'édition manuelle des annotations et de correction des transcriptions proposées par le reconnaiseur
PVAL_3	Valider les transcriptions zone par zone et passer à la zone suivante avec un simple appui sur Entrée
PVAL_4	Pouvoir modifier une annotation fausse en cliquant dessus pour y positionner son curseur et en effectuant ses modifications manuellement
PVAL_5	Indique si les transcriptions ont été fournies manuellement par un humain ou si elles proviennent d'un reconnaiseur
PVAL_6	Faire figurer une fenêtre montrant la page entière découpée en zones avec la zone courante dans une couleur différente
PVAL_7	Valider le travail pour de bon et fermer le document à l'aide d'un bouton prévu à cet effet

Bloc 5 : Lien entre les blocs précédents	
Spécification	Description
LINK_PR_STO	Envoyer les données en entrée vers le système de stockage
LINK_STO_IHM	Extraire les données pour les fournir à l'IHM
LINK_STO_IR	Extraire les données pour les fournir au système de reconnaissance
LINK_IHM_STO	Envoyer les demandes de l'IHM au système de stockage
LINK_IHM_IR	Envoyer les résultats du reconnaiseur vers le système de stockage
LINK_COH	Fournir un logiciel composés de blocs communiquant entre eux de manière fonctionnelle et cohérente

Bloc 6 : Général	
Spécification	Description
GEN_ERGO	Ergonomie de l'application
GEN_ERGO	Concevoir un logiciel évolutif
GEN_ERGO	Fournir un logiciel open source



INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7
Tél. +33 [0] 2 23 23 82 00
Fax +33 [0] 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITE
BRETAGNE
LOIRE

Cti
Commission
des Titres d'Ingénieur

