Rapport de Stage

Développement d'un logiciel de gestion documentaire

Réalisé par

Sacha Lhopital

Sous la direction de

Francis Baril

Année Universitaire 2014-2015

Remerciements

Avant toute chose, il est pour moi indispensable de remercier tous ceux qui m'ont accompagné dans le déroulement de ce stage.

Je commencerais donc par Francis BARIL, directeur Recherche & Développement de DocuLibre Québec. Sa grande disponibilité d'écoute et son implication ont permis à ce stage de se dérouler dans les meilleures conditions possibles. Je remercie également toutes les autres personnes de DocuLibre, pour leur accueil convivial, leur sympathie et leur bonne humeur.

Un merci spécial à Marc DESLANDES, Stéphane MERCIER et l'ensemble du personnel du Cégep de Lévis Lauzon qui m'ont chaleureusement accueilli au Canada.

Je remercie enfin Julie PHILIP, Francis GARCIA, et toutes les autres personnes qui m'ont permis de réaliser ce stage en dehors de la France.

Sommaire

Table des figures

Glossaire

CMIS (Content Management Interoperability Services) : protocole qui a pour but d'augmenter l'interopérabilité entre différents systèmes de gestion de contenu.

Hadoop : framework JAVA Open Source* conçu pour réaliser des traitements sur des volumes de données massifs, de l'ordre de plusieurs pétaoctets (soit plusieurs milliers de To).

Open Source : désigne un logiciel dans lequel le code source est à la disposition du grand public.

REST (Representational State Transfer) : style d'architecture pour les systèmes hypermédia. Ce n'est pas un protocole (tel que HTTP) mais bien un style d'architecture adapté au World Wide Web.

Taxonomie : Structure hiérarchique de classification d'enregistrement.

Zookeeper : logiciel Open Source* de service de configuration et de synchronisation. Zookeeper stocke les données dans un espace de nom hiérarchique, un peu comme un système de fichiers ou une structure de données en forme d'arbre. Les utilisateurs peuvent lire et écrire à partir de/vers ces branches.

Introduction

Les nouvelles technologies ont permis la création d'un nouveau marché pour la gestion, où l'écran permet de gagner du temps par rapport aux supports physiques. Que ce soit dans un cadre personnel ou professionnel, de nombreuses applications se développent pour faciliter cette gestion. Ainsi, pour la plupart des entreprises, il est devenu presque nécessaire d'adopter une de ces applications pour gagner du temps et ainsi rester compétitif.

C'est dans ce contexte que DocuLibre a mis en place un outil d'aide à la gestion, au classement et à l'archivage de données : "Constellio". Mon stage a eu pour objectif de participer au développement de ce logiciel. [1]

Après avoir présenté DocuLibre, je présenterai les informations qui ont été nécessaires au bon départ du projet avec le cahier des charges. Ensuite je détaillerai le fonctionnement précis de l'application créée ainsi que les outils utilisés, ceci dans le rapport technique. Puis je détaillerai un manuel d'utilisation qui répondra aux questions sur l'utilisation du logiciel. Enfin un rapport d'activité dressera un bilan professionnel et personnel du stage dans son ensemble.

1. Ces documents sont la propriété de DocuLibre et sont Confidentiels. Le présent rapport ne peut donc pas être diffusé sans le consentement de ceux-ci.

Chapitre 1

Présentation de DocuLibre

1.1 Création et évolution de DocuLibre

Doculibre Inc. a été créé en 2006 par Rida Benjelloun, Nicolas Belisle et Vincent Dussault, qui la dirigent toujours actuellement. Son activité première est la création de logiciels dans le domaine de la gestion de l'information. Ces logiciels sont basés sur d'autres logiciels libres (Open Source*).

Depuis 2014, l'entreprise entreprend une extension importante, passant progressivement d'une petite entreprise d'environ vingt personnes à une entreprise moyenne d'une cinquantaine de personnes. Ce choix de s'agrandir a pour but de viser un marché qui est à prendre. Ayant déjà conquis le marché francophone avec des clients présents dans plus de cinquante-trois pays, une internationalisation du logicielle est prévue en anglais et en arabe pour poursuivre cette expansion.

L'entreprise compte à l'heure actuelle une cinquantaine de salariés répartis entre Québec, Montréal et Ottawa.

1.2 Activités de Doculibre

Doculibre Inc. propose des solutions permettant de gérer l'information. Grâce à leurs outils, le processus de gestion des documents est grandement simplifié. Ainsi, leur logiciel Constellio, est une plate-forme pouvant accueillir des fichiers représentant des documents (un fichier PDF par exemple) et qui permet d'accéder, de télécharger, de classer, de hiérarchiser et de traiter ces documents.

Pour gérer ces fichiers, trois objectifs principaux servent d'axes au développement de l'application :

La recherche, qui permet aux utilisateurs de Constellio de retrouver l'information quel que soit l'endroit où elle est stockée dans l'application. Ils peuvent faire une recherche classique, ou effectuer des recherches plus avancées selon les paramètres de chaque fichier stocké.

La gestion documentaire et des archives, l'utilisateur peut ainsi organiser ses documents selon sa propre hiérarchie. Contrairement à une gestion documentaire "classique", où changer un dossier de place peut-être difficile, avec leur support informatique il est possible de modifier à tous moment la place des fichiers. L'utilisateur peut également archiver ses documents simplement, via Constellio.

La gestion des collaborateurs, permettant à différentes personnes d'une même entreprise d'avoir accès à certains fichiers ou pas.

Enfin, en plus de ces services techniques, Doculibre propose à ses utilisateurs un service de maintenance et d'assistance en cas de problèmes sur l'application (ex : un problème avec le serveur, un problème de connexion, ou encore une question sur une fonctionnalité).

1.3 Marché et clients

Doculibre Inc. met à disposition ses services et Constellio aussi bien pour les entreprises que pour les organisations.

Le logiciel Constellio est disponible de deux façons différentes pour un client. Il est disponible en téléchargement libre sur le site de l'entreprise, selon le principe de l'Open Source*. L'utilisateur peut alors le télécharger et recevoir les mises à jour gratuitement. Mais un client peut également acheter le logiciel. Dans ce cas, il bénéficie en plus d'un Plan Support qui permet une maintenance en temps réel. En plus de ce Plan Support, le client peut faire des suggestions de fonctionnalités à développer, ce que ne peut pas faire un utilisateur qui s'est procuré Constellio gratuitement.

Parmi les principaux clients on retrouve :

— Bibliothèque et Archives Nationales du Québec (BAnQ),
— Université du Québec à Montréal (UQAM),
— HEC Montréal (Université),
— Secrétariat du Conseil du Trésor (Gouvernement du Québec),
— Ministère du Travail, de l'Emploi et de la Solidarité sociale du Québec (MESS),

1.4 Équipe de développement et partenaires

En ce qui concerne le niveau de qualification du personnel, les équipes de Doculibre sont très variées. Cette variété s'explique par une mixité culturelle au sein même de l'entreprise avec sept nationalités différentes (Canada, Maroc, Italie, France, Brésil, Allemagne, République Dominicaine). Néanmoins, la plupart des développeurs possèdent un équivalent à Bac +2 ou +4 en informatique de gestion.

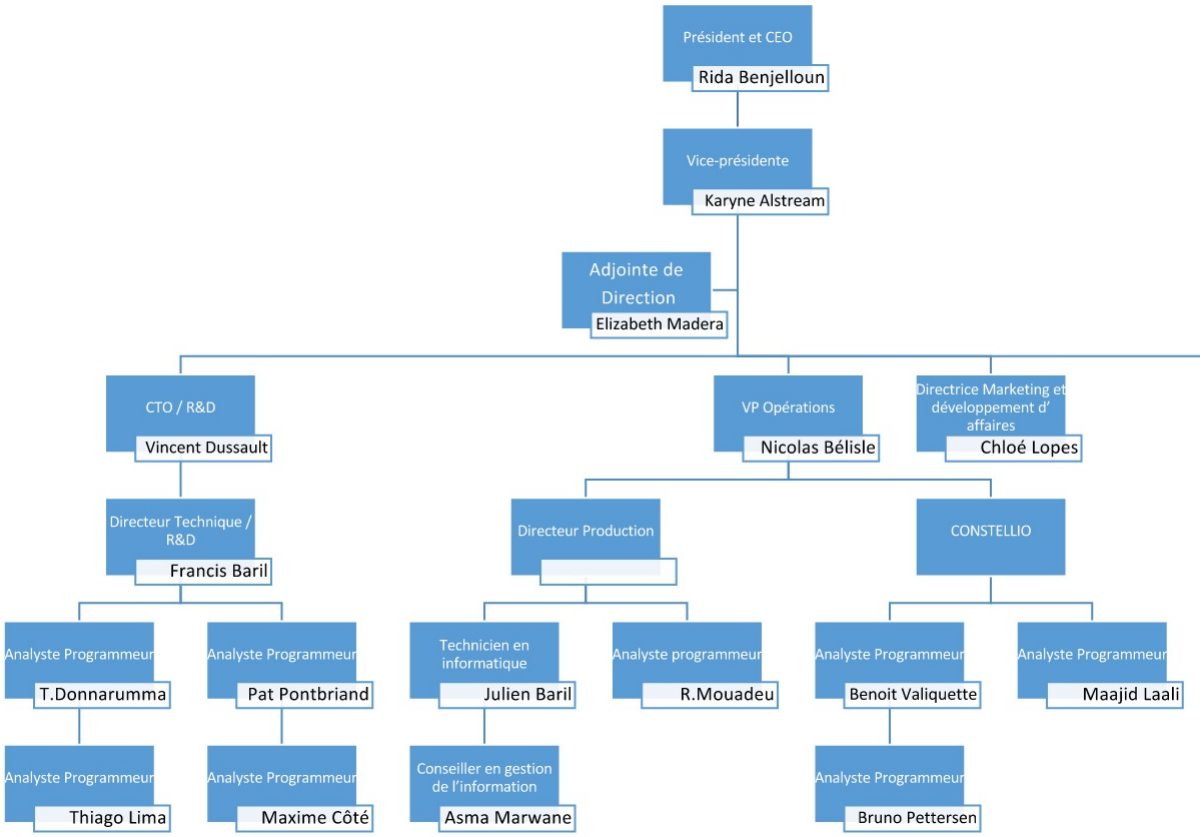
L'équipe de développement est divisée en deux équipes : une équipe à Québec et une autre équipe à Montréal. Cette division permet à l'entreprise de couvrir une plus large zone pour rencontrer les clients, installer le produit, effectuer des démonstrations, etc. En général, les développeurs travaillent souvent sur des fonctionnalités différentes les uns des autres, mais assez fréquemment ils s'entraident et peuvent travailler à deux sur un point particulier. Ce qui les amène souvent a échanger leurs points de vue sur Skype par exemple.

L'entreprise ne possède pas actuellement de partenaires. Pendant les premières années, elle était partenaire avec une entreprise de gestion documentaire qui lui a permis de comprendre les besoins des utilisateurs.

1.5 Organisation et fonctionnement

La figure suivante montre le fonctionnement interne de l'entreprise tel que j'ai pu le percevoir durant mon stage :

Figure 1.1 – Organigramme DocuLibre



L'équipe Constellio utilise deux méthodologies de gestion du projet :

La méthodologie agile SCRUM : La méthodologie agile SCRUM fait partie des méthodologies dites itératives et adaptatives. En effet, elle permet aux développeurs de s'adapter aux besoins du client pour bien répondre à ceux-ci. Ainsi, le client est impliqué tout au long du processus d'implantation, et la validation se fait en continu et non pas à la fin du développement, comme c'est le cas pour la plupart du temps en informatique. Cette méthode permet de réajuster très rapidement et de livrer des solutions conformes aux besoins du client.

La méthodologie Project Management Institute (PMI) : Le PMI est une association professionnelle à but non lucratif qui propose des méthodes de Gestion de projet. Cette méthode découpe le projet en cinq phases :

- L'initialisation où l'équipe définie les tâches à réaliser.
- La planification où on définit l'ordre des tâches à effectuer, les ressources qui seront nécessaires et le temps estimé pour la réalisation.
- L'exécution, phase la plus longue, qui consiste à développer les fonctionnalités définies lors des deux phases précédentes.
- Le suivi qui n'est pas une "phase" à proprement parler, puisqu'elle consiste à suivre le projet après le développement. Il s'agit de bien gérer son implantation un environnement prévu.
- La clôture, au terme d'un suivi plus ou moins long, qui décrète officiellement le projet comme terminé.

En plus de l'application de ces deux méthodes, Constellio est développé en Open Source*. Le choix des fonctionnalités à développer peut alors venir de deux façons différentes :

- La direction reçoit une commande de la part d'un client. Dans ce premier cas, la fonctionnalité, une fois terminée, sera proposée à tous les clients qui possèdent Constellio, comme une mise à jour.
- Le directeur général décide de créer une fonctionnalité. Il faut savoir que les différents employés peuvent tout à fait soumettre leurs propres idées. Dans ce cas, la fonctionnalité sera également accessible à tous les clients qui possèdent déjà le produit.

Dans les deux cas, les différentes personnes concernées se réunissent pour appliquer la méthode SCRUM. Après ces réunions, chacun se voit attribuer une tâche globale (comportant des sous-tâches plus ou moins implicites) avec généralement une durée prévue.

Chapitre 2

Cahier des Charges

L'ensemble de ce chapitre est la propriété de Doculibre Inc.

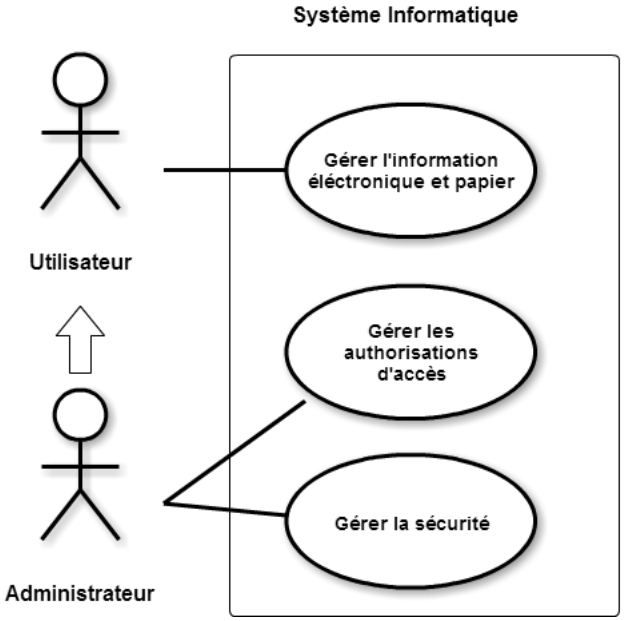
Le cahier des charges actuel de l'application est très complet, car le développement a démarré depuis plus d'un an. De ce fait, uniquement les fonctionnalités notables ou utiles au stage seront évoqués ici.

2.1 Description Générale

Constellio est un logiciel d'aide à la gestion et à la classification de l'information au moyen d'une interface accessible en ligne où une à plusieurs personnes peuvent trier, sélectionner, corriger, etc. un grand nombre de documents.

Les versions de Constellio sont nommées en trois niveaux (par exemple 5.1.14). Le premier niveau est la grande version du logiciel. Le second niveau est augmenté pour chaque nouvelle fonctionnalité (par exemple, l'intégration d'un "bureau"). Le dernier niveau est augmenté lors d'une mise à jour pour corriger un bug ou ajouter une fonctionnalité mineure.

Figure 2.1 – Diagramme de Cas d'Utilisation



Comme le montre le diagramme ci-dessus, l'application doit avant tout permettre de faciliter la gestion de documents pour tous les utilisateurs. L'administrateur du site doit en plus pouvoir gérer précisément les droits d'accès des utilisateurs aux documents et la sécurité de ces informations qui circulent sur l'application.

2.2 Fonctionnalités du Système

Gestion de documents papiers, électroniques, et courriels

Constellio permet la gestion et la classification des documents papiers et électroniques d'une entreprise, ainsi que des courriels ajoutés au système.

Gestion du plan de classification et du cycle de vie des documents

Constellio permet la gestion des documents selon un plan de classification. Le cycle de vie archivistique des documents est également géré, à l'aide d'un module de déclassement.

Sécurité et autorisations

Constellio offre un système de sécurité très granulaire et entièrement personnalisable. Il y est possible, par exemple, de restreindre des autorisations à un certain intervalle de temps, ou de donner une autorisation à un groupe d'utilisateurs. Constellio offre la possibilité de créer des rôles et de leur attribuer des permissions.

Fonctions de bibliothèque avec versions mineures et majeures

Constellio offre la possibilité d'emprunter et retourner des documents pour les modifier. Lors de l'utilisation de ce module, l'application mémorise et gère les différentes versions des documents.

Import et export

Constellio offre l'exportation et l'importation de toutes les données du système. Cette fonctionnalité permet au gestionnaire de produire des sauvegardes de sécurité, ou d'importer des données d'un autre système.

Détection des doublons

Constellio optimise l'espace occupé par le système en ne sauvegardant qu'une seule fois un contenu. En cas de doublon, deux enregistrements pointent vers le même contenu.

Rapports

L'application produit sur demande des rapports complets du système. Le gestionnaire peut demander à imprimer toute la hiérarchie d'enregistrements, selon n'importe quelle taxonomie*, ou encore toute la journalisation.

2.3 Exigences non-fonctionnelles

2.3.1 Spécifications techniques

Constellio est une application développée en JAVA avec une Base de donnée SolR et une interface WEB intégrée avec VAADIN. L'interface web de l'application impose de fortes contraintes au projet. En effet, il faut gérer les différents navigateurs qui existent. La connexion Internet est aussi une exigence pour l'application. Cette connexion est nécessaire à la bonne interaction avec la base de données et au bon fonctionnement de l'application. Sans connexion, l'application ne pourra pas fonctionner.

Tous ces outils sont open source*.

2.3.2 Documentation du projet

Le projet est documenté de deux façons différentes, mais qui se complètent.

Dans un premier temps, le code de programmation de l'application est rédigé selon les règles du *Clean Code*[1] de façon à ce que chacun puisse comprendre le fonctionnement du code.

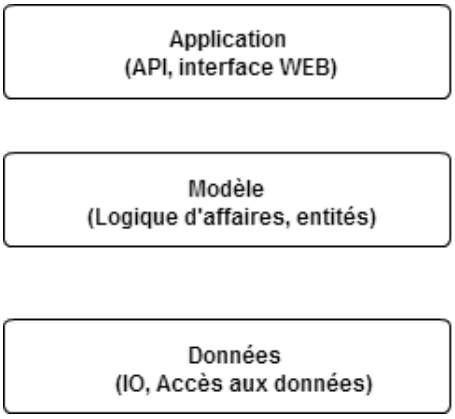
Dans un second temps, une documentation complète expliquant le fonctionnement de l'ensemble du projet, son but et comment il a été réalisé, a été prévu et est régulièrement mis à jour.

1. cf. bibliographie

2.4 Couches Logicielles

Constellio est divisé en trois couches logicielles :

Figure 2.2 – Couches Logicielles



Couche applicative

La couche applicative comprend :

— l'API de communication avec les serveurs : Services REST*, Services CMIS* et Requêtes Solr avec sécurité des documents.
— Interface utilisateur.
— Les modules de gestion des utilisateurs et de recherche.
— la configuration de base du logiciel.
— la mise à jour automatique.

Couche modèle

La couche modèle comprend :

— La gestion de la sécurité.
— Les taxonomies*.
— Les traitements en lot.
— Mécanisme de gestion de conflits automatique (en cas de doublons).
— API de recherches.

Couche Données

La couche données comprend :

— Services d'Input/Output : détection de la mauvaise fermeture des flux de données et détection de la mauvaise suppression si des fichiers temporaires existent.
— Service de lecture/écriture dans Solr.
— Services de sauvegarde de contenu dans un serveur externe à SolR : Hadoop*.
— Services de sauvegarde des configurations de l'application dans Zookeeper* et dans Hadoop*.

Chapitre 3

Rapport Technique

Dans cette partie on détaillera le fonctionnement de l'application au travers des différentes tâches que j'ai réalisé pendant le stage. Dans la partie précédente on a pu voir toutes les exigences développées avant mon arrivée. Ayant réalisé principalement des tests, ceux-ci concernent les fonctionnalités précédemment évoquées.

3.1 Conception

Pendant la réalisation du stage, Sélénium, un outil open source* qui permet de tester l'interface utilisateur des applications Web, était utilisé par l'ensemble de l'équipe. C'est donc tout naturellement que j'ai réalisé mes tests avec celui-ci.

Mise à part cet outil, une fonctionnalité est généralement développée en TDD (cf. 3.1.2).

3.1.1 Présentation du fonctionnement de Sélénium

Dans le cadre du développement d'une application d'une grande envergure comme Constellio, les tests sont indispensables pour son évolution, et prennent une part non-négligeable du développement.

Sélénium est un de ces outils d'automatisation, concernant les tests d'interface des applications Web. Il se compose de deux parties :

— Selenium IDE : c'est une extension de Firefox, qui permet d'enregistrer une suite d'actions, qu'il sera possible de rejouer à volonté ;
— Selenium WebDriver : il s'agit cette fois d'une API pour JAVA, permettant de programmer des actions sur l'interface, et de vérifier les réponses. Les actions à réaliser peuvent être des extensions depuis Sélénium IDE.

Pour plus de détails sur son fonctionnement, se référer à la section 3.2

3.1.2 Présentation de la méthode Test Driven Development

Le Test Driven Development (TDD) ou en français "développement piloté par les tests" est une technique de développement qui préconise, avant même d'écrire une ligne de code, de créer un test d'acceptation pour tester l'ensemble de la fonctionnalité à développer et un test unitaire pour chaque composante de celle-ci.

Le cycle préconisé par TDD comporte cinq étapes :

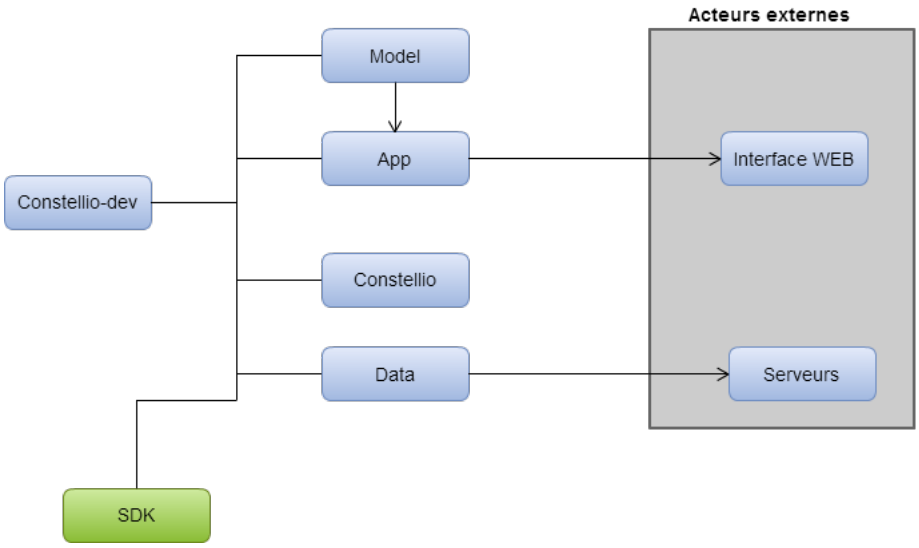
— écrire un premier test ;
— vérifier qu'il échoue (car le code qu'il teste n'existe pas), afin de vérifier que le test est valide ;
— écrire juste le code suffisant pour passer le test ;
— vérifier que le test passe ;
— enfin réusiner le code, c'est-à-dire l'améliorer tout en gardant les mêmes fonctionnalités.

Il est obligatoire de réussir une formation TDD en vingt étapes avant de se joindre à l'équipe de développement de Constellio. Cette formation permet l'apprentissage des frameworks JUnit et Mockito, des concepts de chaque technique, et des normes de programmation propres à l'équipe.

3.1.3 Arborescence des fichiers de l'application

Comme indiqué dans le cahier des charges, l'application est divisée en trois couches et par différents packages (cf. figure 3.1).

Figure 3.1 – Arborescence des fichiers



App contient l'ensemble des API du système, les interfaces de configuration des fonctionnalités, le programme de lancement de l'application et les fichiers relatifs à l'interface WEB.

Constellio contient l'ensemble des ressources externes nécessaires à l'application : des fichiers de configurations pour les langues, des fichiers pour faire le lien entre application et base de données, etc.

Data est un répertoire particulier qui contient entre autres le code permettant de récupérer et d'envoyer les données sur le serveur.

Model contient un ensemble de classes pour l'implantation de tout ce qui est développé dans App. Model contient également certaines extensions, des normes de configurations, etc. On y retrouve par exemple : les classes permettant la localisation des fichiers dans l'application ; Les dépendances relatives au calculs automatiques ; etc.

Sdk enfin, constitue l'ensemble des classes de tests de l'application. Ces tests concernent toutes les couches précédemment énoncées. Sdk possède également une version de démonstration avec tout un jeu de donnée permettant la validation de ces tests.

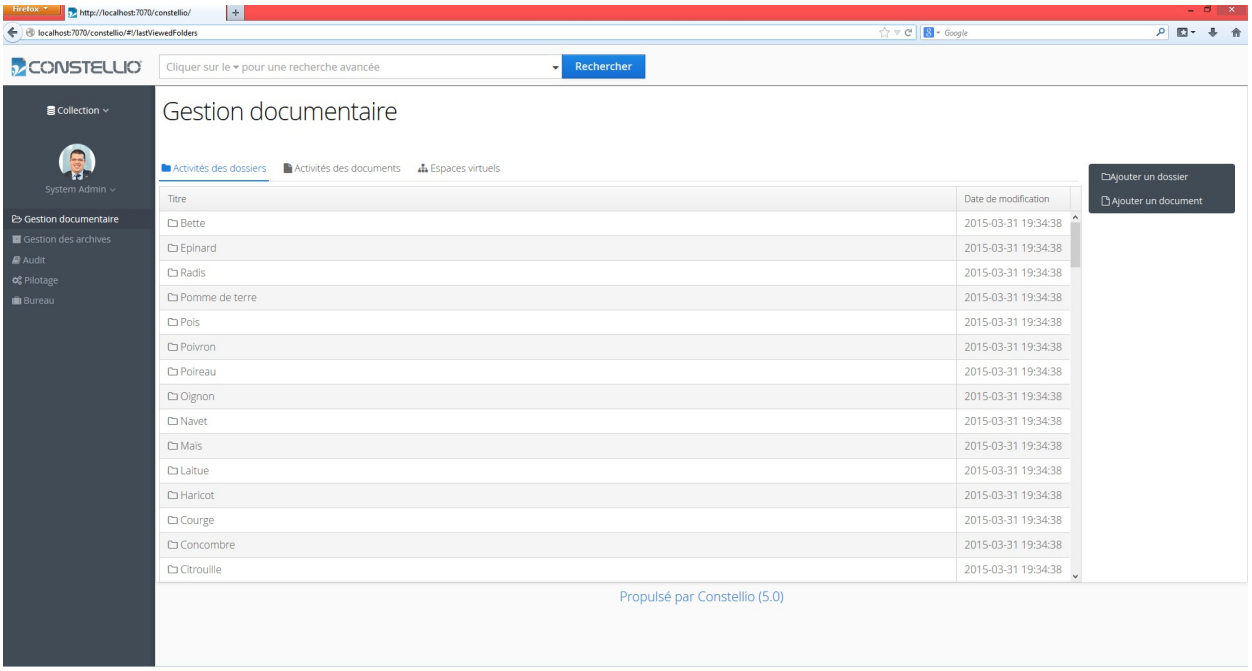
3.2 Résultats

3.2.1 User Interface Smoke Tests

Demande du tuteur : Tester l'interface de l'application.

Le visuel suivant, tiré de l'application, donne un exemple de l'ensemble des interfaces à tester.

Figure 3.2 – Page d'accueil



Par exemple pour cet écran, il faut tour à tour cliquer sur les boutons du menu principal à gauche et vérifier qu'ils amènent à la bonne vue. Puis il faut vérifier qu'en cliquant sur le menu dynamique au milieu de la page, on obtient également le bon affichage. Le but est de tester les boutons auxquels l'utilisateur a accès et le bon affichage correspondant.

Naviguer d'une page à l'autre

Une des règles les plus importantes sur les tests d'une application est qu'ils doivent tous être indépendants les uns des autres. Cela pour éviter qu'en cas d'échec d'un des tests ne se produise un effet boule de neige sur les autres tests. Ainsi, entre chaque écran à tester, il faut naviguer entre les pages automatiquement, pour pouvoir tester les comportements manuels les uns après les autres.

Pour ce faire, nous simulerons les cliques de souris d'un utilisateur pour voir si l'interface réagit bien et affiche le bon résultat. La navigation s'effectue simplement, grâce à une fonction de Sélénium *navigateTo()* :

```
1 driver.navigateTo().url(NavigatorConfigurationService.RECORDS_MANAGEMENT);
```

Ici, *driver* correspond au navigateur courant de Firefox. *NavigatorConfigurationService* est une classe sérialisée qui regroupe toutes les urls de l'applications. Ici on demande au navigateur de naviguer automatiquement jusqu'à la page de gestion des dossiers.

Sélectionner le bouton et simuler le clique

Après avoir navigué jusqu'à la bonne vue, il faut sélectionner un élément de la page courante pour interagir avec lui.

Si on veut cliquer sur le bouton "Ajouter un Document" du menu en haut à droite par exemple (cf. 3.2), il faut sélectionner ce bouton en passant par le code HTML (son nom de classe, son identifiant, son tagName, etc...).

```
    public ConstellioWebElement getAddDocumentButton() {
        return driver.findElement(By.className("v-button-addDocument"));
    }
```

Ainsi, la sous-fonction précédente sélectionne dans *element* l'élément dont le nom de classe contient "v-button-addDocument". Ensuite on peut cliquer dessus avec la fonction *click()* de Sélénium.

Un problème est survenu rapidement : le *click()* ne fonctionne pas sur certains postes. Enfaîte, certains postes de l'équipe fonctionnent sous Windows 8, d'autres sur LinuX et d'autres encore avec Mac. La fonction ne marche pas sur Windows 8.

Par défaut, la taille d'affichage du système d'exploitation est réglée à 150/160%. Or, Sélénium pour utiliser *click()* fonctionne comme suit : il sélectionne l'élément sur la page et mémorise sa position (basé sur un affichage de la page à 100%) puis il clique dessus. Si l'affichage est réglé à 150%, Sélénium n'arrive pas à cliquer au bon endroit. Il a suffi de modifier les paramètres de Windows des postes correspondant pour forcer un affichage à 100%.

Une fois tous les problèmes résolus : en utilisant la fonction *getAddDocumentButton* précédente, on obtient le test fonctionnel suivant :

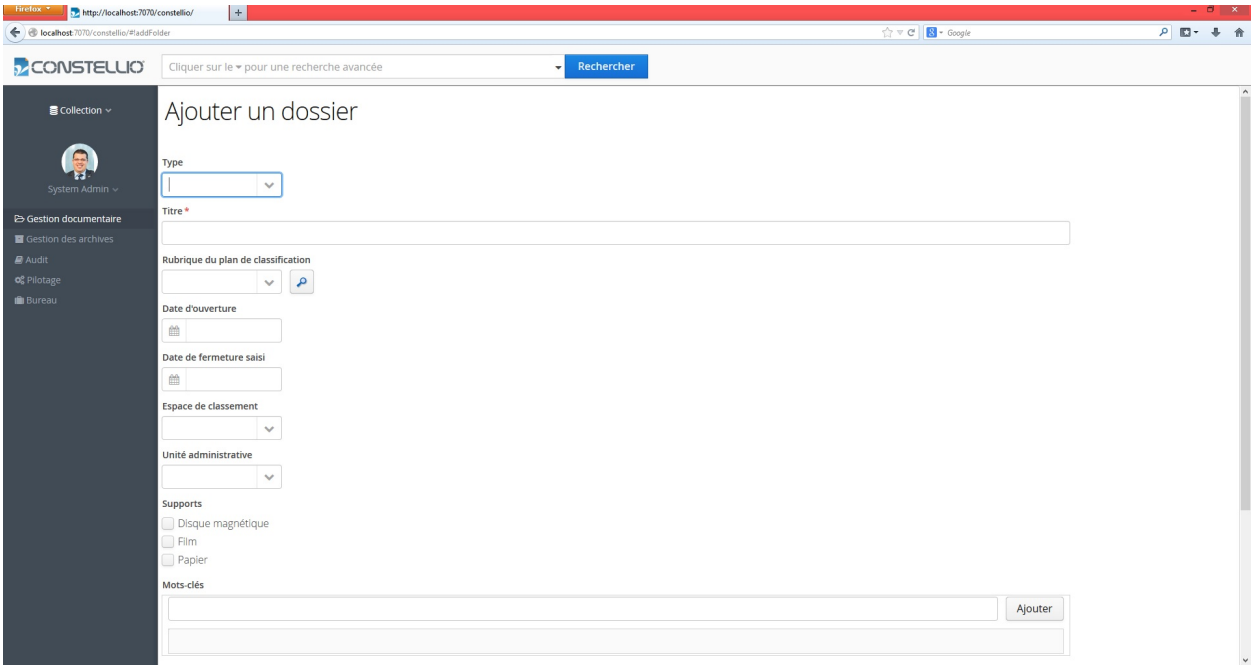
```
public void givenRecordsManagement[When]ThenInLastViewedFolders()
    throws Exception {
    //navigation
    driver.navigateTo()
        .url(NavigatorConfigurationService.RECORDS_MANAGEMENT);

    //clique sur le bon element
    driver.getAddDocumentButton().click();
    //attends que la nouvelle page se charge
    driver.waitForPageReload();
    //verifie que la page courante correspond au bon ecran
    assertThat(driver.getCurrentPage())
        .isEqualTo(NavigatorConfigurationService.ADD_DOCUMENT);

    //Retourne en arriere et verifie le retour.
    driver.getCancelButton().click();
    driver.waitForPageReload();
    assertThat(driver.getCurrentPage())
        .isEqualTo(NavigatorConfigurationService.RECORDS_MANAGEMENT
            + "/lastViewedFolders");
}
```

3.2.2 Tests création d'un dossier

Demande du tuteur : Tester la bonne création d'un fichier en passant par l'interface utilisateur.

Il faut compléter tous les champs du formulaire de création d'un dossier et voir si chaque information est correctement sauvegardée.

Figure 3.3 – Formulaire de création d'un dossier



C'est le même schéma que pour les User Interface Smoke Tests : il faut sélectionner le champ à compléter, le compléter, cliquer sur un bouton de validation et vérifier.

Intéraction entre deux champs : Espace de classement et Unité Administrative

Pendant ces tests, l'interaction entre ces deux champs est apparue comme anormale. En effet ces deux conditions sont liées l'une à l'autre : une unité administrative peut contenir plusieurs espaces de classement ; Un espace de classement peut correspondre à plusieurs unité administrative.

Par exemple, si l'unité administrative "10" ne contient que l'espace de classement "A", et que cet espace "A" n'est contenu que dans l'unité "10" : si l'utilisateur sélectionne l'espace A, l'unité administrative doit forcément être le numéro 10. S'il sélectionne l'unité 10 avant l'espace de classement, l'espace sélectionné sera automatiquement le A.

Voici donc le cas qui pose problème : si l'utilisateur sélectionne l'espace de classement A (ou l'unité administrative 10) par erreur dans le formulaire, il ne peut plus changer son choix. En effet, comme l'unité 10 est sélectionnée, il ne peut que choisir l'espace de classement A, et inversement.

```
/** Condition du champs espace de classement */
if (filingSpaceField != null && filingSpaceField.getFieldValue() != null) {

    String currentFilingSpaceValue = filingSpaceField.getFieldValue();

    List<String> administrativeUnitsForFilingSpace =
        getAdministrativeUnitsForFilingSpace(currentFilingSpaceValue);

    availableOptions = (List<String>) ListUtils.retainAll
            (availableOptions, administrativeUnitsForFilingSpace);
}
```

```
/** Condition du champs unite administrative */
if (administrativeUnitField != null
    && administrativeUnitField.getFieldValue() != null) {

    String currentAdminUnitValue =
        administrativeUnitField.getFieldValue();

    List<String> filingSpacesForAdministrativeUnit =
        getFilingSpacesForAdministrativeUnit(currentAdminUnitValue);

    availableOptions = (List<String>) ListUtils.retainAll
        (availableOptions, filingSpacesForAdministrativeUnit);
}
```

Les deux codes ci-dessus représentent les conditions actuellement appliquées aux deux champs.

La première condition (cf. premier extrait de code) agit lorsque l'on complète le champ espace de classement. Si le champ espace de classement a été complété par l'utilisateur (ligne 2), on récupère la valeur saisie (ligne 4) et on récupère la liste des unités administratives possibles en fonction de l'espace de classement. Plus tard dans la fonction, est affiché l'ensemble des valeurs que l'utilisateur peut sélectionner (*availableOptions* ligne 9).

Le second code, qui intervient lorsque l'on complète le champ unité administrative fait présentement la même chose mais dans l'autre sens.

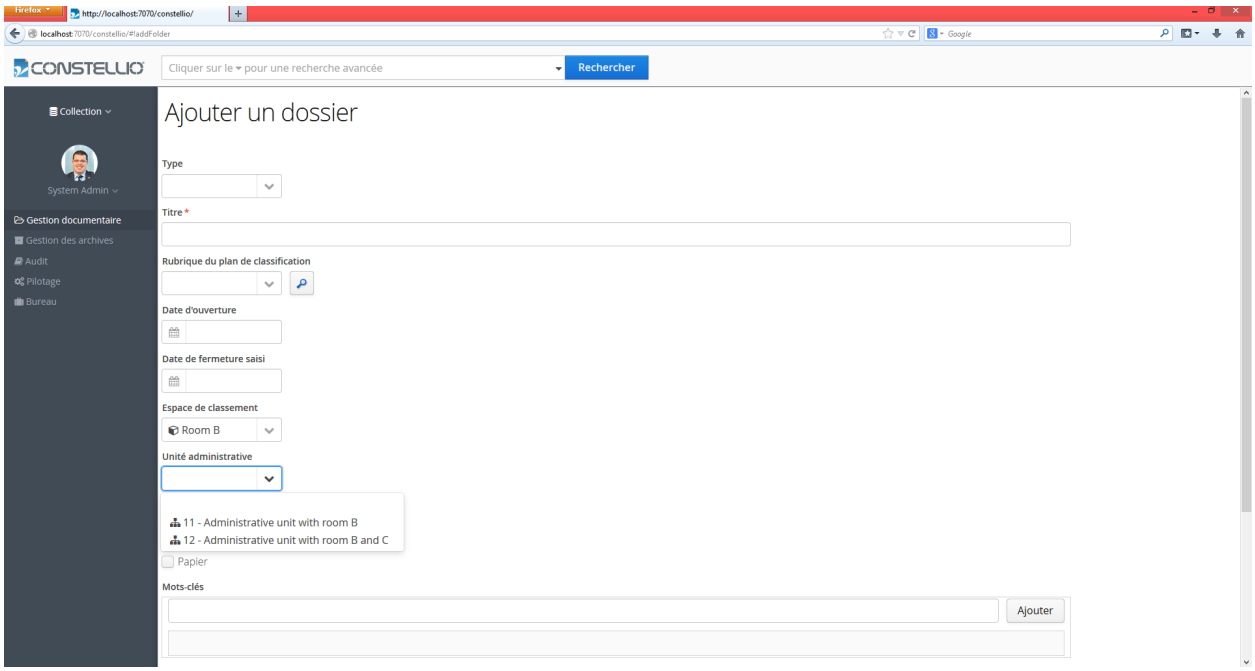
D'après la logique énoncée précédemment, une de ces conditions est en trop. Il suffit juste d'enlever une de ces deux conditions puisqu'elles se contredisent. C'est la condition du premier code qui a été supprimé. Ainsi l'unité administrative dépend de l'espace de classement choisi.

Ergonomie

Visuellement dans le formulaire (cf. 3.3), l'utilisateur doit choisir l'unité administrative avant de sélectionner l'espace de classement alors que la modification précédente entraîne une dépendance inverse. Du coup, le formulaire n'est plus assez commode pour l'utilisateur. Il faut donc réarranger l'ordre du code pour que le formulaire s'affiche de façon plus ergonomique.

Cette modification intervient sur une classe qui se concentre sur l'affichage des champs. Cette classe utilise la classe dans laquelle nous sommes intervenus lors du dernier problème.

Figure 3.4 – Formulaire de création d'un dossier - Résultat



Comme le montre la figure 3.4, en changeant l'ordre d'affichage des deux champs, l'utilisateur va naturellement constater visuellement une modification de l'unité administrative s'il change l'espace de classement (car l'oeil descends naturellement vers le bas). De plus, avec cet ordre d'affichage, l'utilisateur va naturellement avoir tendance à sélectionner l'espace de classement en premier.

3.2.3 Règles de gestions d'un document et affichages

Demande du tuteur : Tester que le menu d'un document s'affiche correctement. Les bonnes options doivent afficher (modifier, ajouter, supprimer, etc.) en fonction des permissions de l'utilisateur (read, create, write, delete) et en fonction de l'état du document (détruit, actif, occupé, etc.).

Pour réaliser cette mission, les fonctionnalités ont été réalisé avec la méthode TDD : en même temps que ces tests, il a ainsi été modifié l'affichage pour que le menu s'affiche de façon contextuel par rapport au statut du dossier.

Nous détaillerons plus particulièrement la gestion de l'emprunt et du retour d'un document.

De base dans l'application, sans modification par un administrateur, un utilisateur peut avoir trois rôles distincts :

— User
— Manager
— Responsable de la Gestion Documentaire (RGD)

Les actions disponibles en fonction du rôle de l'utilisateur sont détaillées ci-après (3.5).

Le compte d'un utilisateur, possède un rôle de base. Mais par la suite, une action spécifique (consulter, écrire, créer, supprimer) peut lui être attribuée pour un espace de classement précis ou une unité administrative. Par exemple : l'utilisateur X est un Manager, mais un administrateur lui a restreint ces droits de sorte qu'il ne peut que consulter les documents présents dans l'unité administrative 10.

Figure 3.5 – Droits des utilisateurs d'après leur rôle

Documents

	U	M	RGD
Partager un document	☐	☑	☑
Créer un document	☑	☑	☑
Modifier un document	☑	☑	☑
Effacer un document	☑	☑	☑
Retourner les emprunts des autres	☐	☐	☑

De base dans l'application, sans modification par un administrateur, un dossier peut avoir quatre statuts :

— Actif : Le fichier peut être consulté, édité et supprimé librement par tous les utilisateurs en ayant le droit.
— Semi-Actif : Le fichier ne peut être que consulté par un User ou un Manager. Mais un RGD peut le modifier.
— Versés : Le fichier ne peut plus être consulté, édité ou supprimé par un User ou un Manager. Seul un RGD peut le supprimer (et uniquement le supprimer).
— Détruit : Le fichier n'est plus accessible et est considéré comme définitivement supprimé pour tous les rôles.

Compte tenu des règles de gestion des rôles, et en tenant compte du statut du dossier, les fonctions d'emprunt d'un document sont énumérées telles que pour un document Actif ou Semi-Actif :

1. Si je peux écrire, le bouton emprunter est disponible (à condition que le document ne soit pas déjà emprunté).
2. Si j'emprunte, le bouton retourner est disponible.
3. Si j'ai le droit de retourner le document des autres, le bouton retourner est disponible (à condition que je puisse écrire spécifiquement dans l'espace de classement du document et que celui-ci soit emprunté).

1. Si je peux écrire, le bouton emprunter est disponible

```
1 actionsComponent.setCheckOutButtonState(getCheckOutState());
```

Premièrement, pour modifier l'affichage du bouton "CheckOut" (= bouton emptunter), on définit dans le menu d'affichage *actionsComponent* un état pour ce bouton. Ainsi, il suffit de récupérer dans la fonction *getCheckOutState()*, l'état dans lequel doit être le bouton pour modifier l'affichage. Détaillons cette méthode ci-dessous :

```
1 private ComponentState getCheckOutState() {
      if (authorization.canWrite(getCurrentUser(), getRecord())) {
3         if (isCheckOutPossible()) {
              return ComponentState.ENABLED;
5         }
          return ComponentState.DISABLED;
7     }
      return ComponentState.INVISIBLE;
9 }
```

Cette méthode retourne un état défini par la classe énumérée *ComponentState* qui peut-être : *ENABLED(true, true), DISABLED(true, false) ou INVISIBLE(false, false)*. Ces différents états correspondent au statut du bouton : le premier boolean indique s'il est visible et le second s'il est actif (Si le bouton est inactif, il sera légèrement grisé et inaccessible pour l'utilisateur).

Ligne 1, *authorization.canWrite(...)* est une méthode qui retourne vrai si l'utilisateur passé en paramètre à les droits d'écriture sur un enregistrement donné. Si c'est le cas et que l'enregistrement est empruntable (ligne 2), alors le bouton emprunter est dans un statut *ENABLED*. Si l'utilisateur peut emprunter, mais que le fichier est déjà emprunté, le bouton sera *DISALED*. Enfin si l'utilisateur ne peut pas emprunter de base, le bouton sera *INVISIBLE*.

2. Si j'emprunte le bouton retourner est disponible.

Pour ce qui est du bouton "CheckIn" (= bouton retourner, voir code ci-dessous), si l'utilisateur à les droits en écriture sur l'enregistrement, et si le retour est possible, alors le bouton est *ENABLED*. Par contre s'il n'a pas les droits, le bouton sera *INVISIBLE*. Le bouton retourner ne peut pas être *DISABLED* car cela ne ferais aucun sens si le bouton "CheckOut" était visible, d'avoir le bouton retourner inactif.

```
1 private ComponentState getCheckInState() {
      if (authorization.canWrite(getCurrentUser(), getRecord())) {
3         if (isCheckInPossible()) {
              return ComponentState.ENABLED;
5         }
      }
7     return ComponentState.INVISIBLE;
  }
```

3. Si je peux retourner le document des autres, le bouton retourner est disponible

Enfin, pour rajouter cette dernière condition, il suffit de rajouter une condition spéciale dans le code précédent sur la ligne 3. En écrivant :

```
  if(isCheckInPossible()
2 || (getCurrentUser().has(Permission.RETURN_OTHER_USERS_DOCUMENTS)
      && contentIsCheckedOut())) {
4     return ComponentState.ENABLED;
  }
```

Ainsi : si le retour du document est possible OU si l'utilisateur possède la permission de retourner l'emprunt des autres utilisateurs et que le document est emprunté par quelqu'un d'autre ; Alors le bouton sera actif et visible.

3.3 Perspectives

Concernant l'ensemble des tests que j'ai réalisé, de nombreux points sont améliorables.

Tout d'abord, et ce, malgré une volonté de bien faire, la propreté du code est nettement améliorable tant du point de vue de la lisibilité que du point de vue structurel.

Ensuite, après réflexions avec mon tuteur, il peut apparaître avantageux d'ajouter des tests encore plus précis que ceux que j'ai réalisés. Cette envie de rajout est loin d'être nécessaire, car les tests que j'ai réalisés sont déjà une excellente base de ce que l'utilisateur attend de l'application. Néanmoins, cette idée n'est pas à mettre de côté, car une application qui possède un très grand nombre de tests pertinents a plus de chance de correspondre aux besoins de l'utilisateur.

Enfin, ma formation n'étant pas terminée, et n'ayant côtoyé l'application que pour quelques semaines, mon code peut-être améliorable. Certaines fonctions sont sans doute mal utilisées et mal optimisées.

Chapitre 4

Manuel d'Utilisation

4.1 Manuel d'installation

À l'heure actuelle, Constellio n'est pas encore disponible. Il est donc impossible de définir un manuel d'installation de l'application que ce soit pour obtenir les codes sources gratuitement ou en installant le logiciel comme client.

4.2 Manuel d'utilisation

Compte tenu du grand nombres de fonctionnalités de l'application, ne seront détaillées ici que les fonctionnalités étudiés pendant le stage.

Pour effectuer les étapes suivantes, l'utilisateur doit impérativement se connecter : Dans l'espace prévue à cet effet (cf. figure 4.1) mettre dans la case login et mot de passe le login et le mot de passe correspondant à votre compte. Puis cliquez sur "Se Connecter". En cochant la case "Se souvenir de moi", votre ordinateur mémorisera vos identifiants lors de la prochaine connexion.

Une fois connecté, vous pouvez librement accéder aux fonctionnalités décrites ci-dessous, dans la mesure de vos permissions.

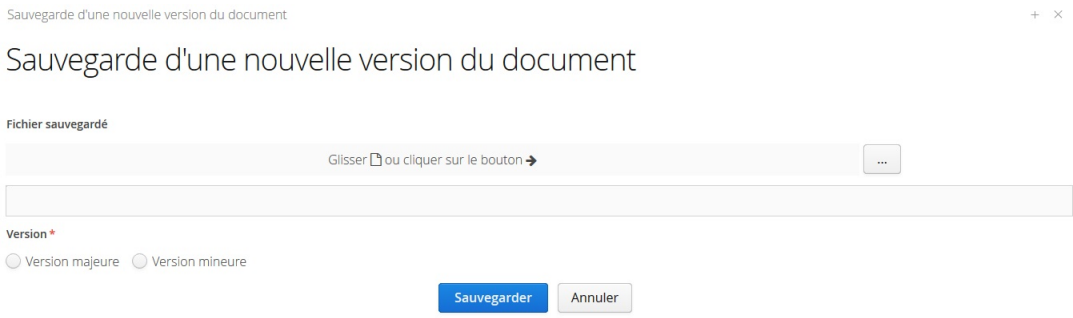
Figure 4.1 – Formulaire de connexion

- **Créer un Dossier - Créer un Document :**
  1. Une fois connecté, au centre du site est apparue la liste des dossiers récemment en activité. Cliquez sur "Ajouter un Dossier" ou "Ajouter un Document" dans le menu contextuel de droite.
  2. Complétez le formulaire ainsi apparu : attention tous les champs Titre, Rubrique du Plan de Classification, Date d'ouverture, Espace de Classement et Unité Administrative sont obligatoires.
  3. Cliquez sur "Sauvegarder".

Figure 4.2 – Formulaire d'ajout d'un dossier

- **Emprunter - Retourner un Document**
  1. Une fois que vous avez choisi le document à emprunter - retourner, cliquez dessus pour accéder à sa vue de détail. Cliquez sur "Emprunter" - "Retourner" dans le menu contextuel de droite.
  2. Dans le cas du retour : une nouvelle fenêtre s'ouvre pour vous permettre de mettre en ligne la nouvelle version du document. Il faut aussi préciser si la nouvelle version du document est majeure ou mineure.
  3. Cliquez sur "Sauvegarder".

Figure 4.3 – Formulaire de retour d'un document



- **Effectuer une Recherche Avancée**
  1. Dans la barre de recherche en en-tête, cliquez sur la petite flèche noire pour afficher le panel de recherche avancée.
  2. Séléctionnez un Type pour la recherche : soit Document, soit Contenant, soit Dossier.
  3. Pour sélectionner un critère de recherche particulier, déroulez les options d'un des champs de sélection. Choisissez l(es)'option(s).
  4. Cliquez sur le bouton "Rechercher", en bas à gauche du formulaire de recherche avancée.

Figure 4.4 – Formulaire de recherche avancée

Chapitre 5

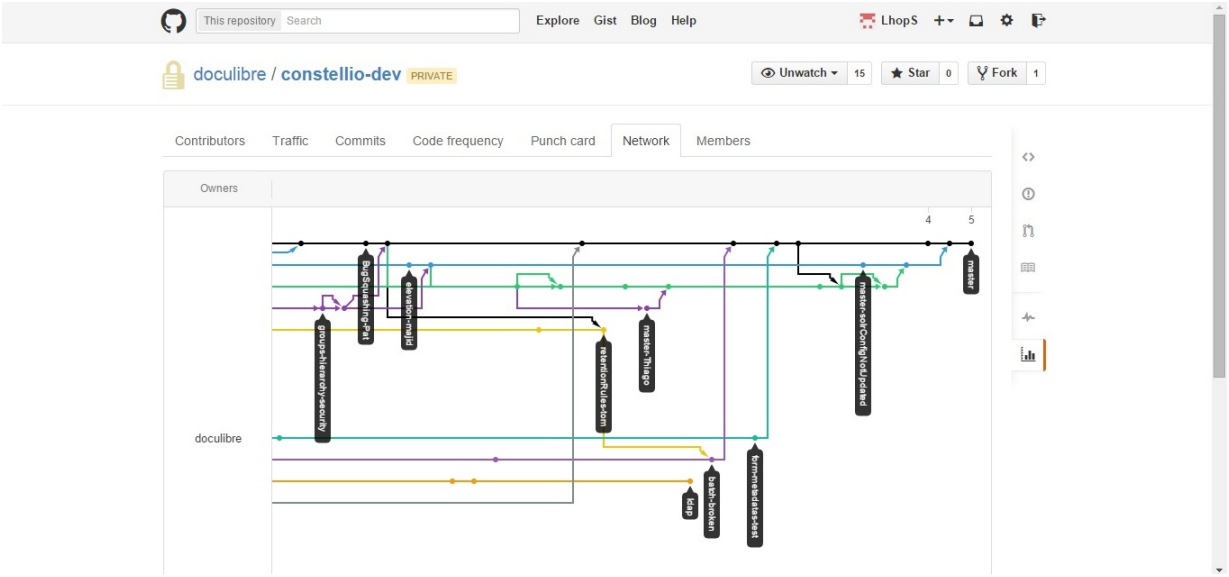
Rapport d'Activité

5.1 Présentation des outils utilisés

5.1.1 GitHub

GitHub est un service WEB d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Ce site propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres.

Figure 5.1 – Interface GitHub



L'ensemble des documents de Constellio sont sur GitHub, en privé pour le moment. GitHub est utilisé par l'équipe pour gérer les différentes versions du logiciel et, à terme, la plate-forme permettra aux autres développeurs de se tenir informer de l'évolution du projet quand il sera en Open Source*.

5.1.2 PhantomJS

PhantomJS est un script, un navigateur utilisé pour automatiser les interactions d'une page Web. PhantomJS fournit une API en JavaScript permettant la navigation automatisée, la capture d'écran et la simulation du comportement des utilisateurs qui en fait un outil commun utilisé pour exécuter des tests unitaires basés sur navigateur. PhantomJS est basé sur Webkit qui en fait un environnement de navigation similaire à Safari et Google Chrome. C'est un logiciel open-source.

L'ensemble des tests de Sélénium sont développés avec Firefox. Quand le test roule, on peut constater les différents cliques de Sélénium sur le navigateur, ce qui est bien pratique.

Par contre, lorsque tous les tests sont roulés sur le serveur d'intégration de l'application, PhantomJS est utilisé à la place de firefox. Dans ce cas, on ne voit pas visuellement le test fonctionner, mais le test est beaucoup plus rapide.

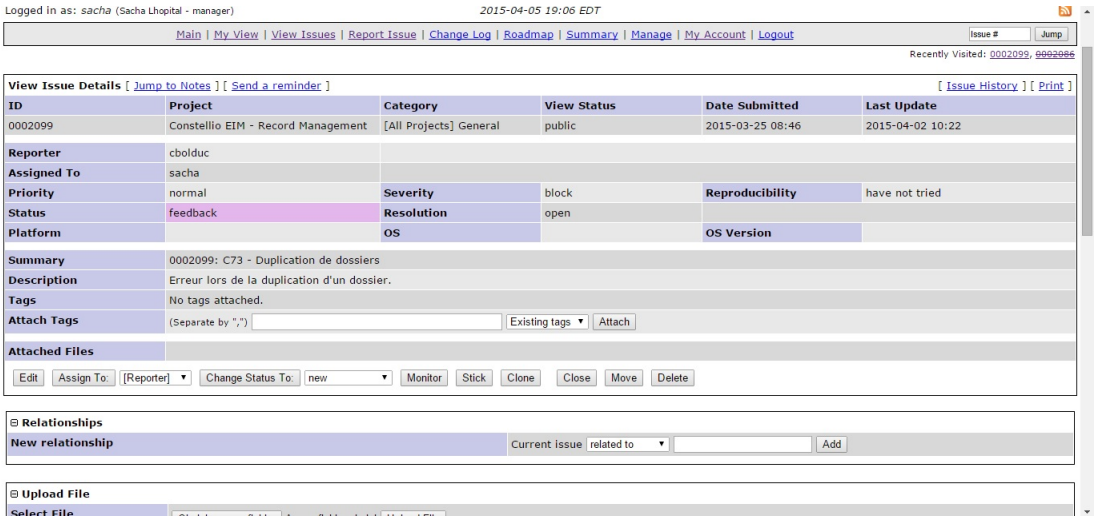
5.1.3 Mantis Bug Tracker

Mantis est un système de suivi de bugs basé sur une interface web.

Le principe de cet outil consiste à enregistrer la déclaration d'un bug, puis pour les développeurs concernés, à mettre à jour l'avancement de sa résolution, jusqu'à sa clôture. Le déclarant de l'anomalie peut s'informer à tout moment de l'avancement du traitement de son problème.

L'ensemble de l'équipe de DocuLibre utilise cet outil pour pouvoir facilement trouver les bugs, les traiter par ordre de priorité et les assigner à la personne la plus à même de les résoudre. Cet outil permet ainsi un gain de temps considérable et une communication rapide.

Figure 5.2 – Interface Mantis Bug Tracker



5.2 Cycle de développement

Les cycles de développement du projet étaient rythmés par les Daily-Meeting pour discuter du projet. Chaque nouvelle réunion était l'occasion d'un nouvel échange entre développeurs et ce fonctionnement a permis de mesurer l'évolution du projet pas à pas.

De plus, chaque Sprint de deux semaines (puis une semaine par la suite) permettait d'effectuer un travail conséquent sur l'application, sans être pour autant trop long.

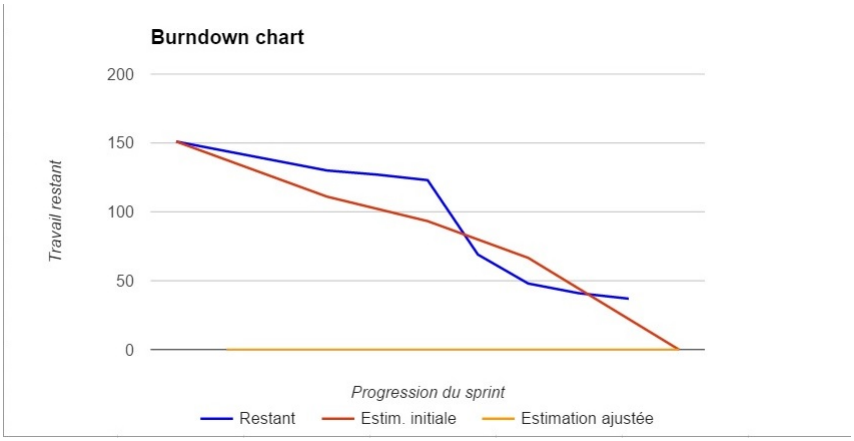
Les dernières semaines du stage, peu avant la mise sur le marché de Constellio, les sprints ont été raccourcis à une semaine car l'équipe faisait surtout du traitement de bugs et presque plus de développement de fonctionnalités.

Figure 5.3 – Tâches du premier sprint

Jour de semaine		L	M	M	J	V	L	M	M	J	V
Jour	0	1	2	3	4	5	6	7	8	9	10
Man power	0,(	3	3	3	2	2	3	3	5	5	5
Borderow deposit	3	3	3	3	3	3	3	3	3	3	
User status	7	7	7	7	7	7	7	7	7	7	
Fix UI tests run time	4	0	0	0	0	0	0	0	0	0	
Decommissioning tests (back-end + UI)	21	21	21	21	21	21	0	0	0	0	
Add roles page tests	4	4	4	4	4	4	0	0	0	0	
Schema/Metadata tests	35	35	35	35	35	35	28	21	14	10	
Global users/groups	7	7	7	7	7	7	7	0	0	0	
Taxonomy page tests	7	7	7	7	7	7	7	7	7	7	
UI Smoke tests	21	21	21	21	21	21	7	0	0	0	
Edit profile (mighty checkbox of doom)	21	21	14	7	4	0	0	0	0	0	
Make integration server build!	7	4	4	4	4	4	0	0	0	0	
Document decommissioning	14	14	14	14	14	14	10	10	10	10	

Néanmoins, chaque sprint était l'occasion de faire une rétrospective sur le travail de l'équipe. Un burndown chart (graphique représentant l'évolution de la quantité de travail restante par rapport au temps global du sprint) est réalisé pendant le sprint (exemple figure 5.4), pour permettre à l'équipe de mesurer son efficacité, jour après jour et de s'adapter au fur et à mesure. Ci-dessous se trouve le burndown chart du premier sprint auquel j'ai participé. Le tableau un peu plus haut (figure 5.3) donne les tâches prévues pour le sprint en cours avec leurs efforts estimés en heure de travail. Par exemple, les UI Smoke tests étaient évalués à 21h de développement. Mais finalement le test a duré plus de trois jours pour se terminer au bout d'une semaine.

Figure 5.4 – Exemple d'un burndown chart



Sur le graphique, l'axe des ordonnées représente les quatorze jours du sprint. L'axe des abscisses représente l'effort évalué en nombre d'heures (pour ce sprint, 150h au départ). La courbe bleu indique l'effort restant estimé et la rouge, l'estimation initiale. Comme l'indique ce graphique, la différence entre la courbe bleue et la rouge du début du sprint jusqu'à sept jours environ correspond aux retards pris sur les UI Smoke tests. Sans ce retard de quelques jours, la courbe bleue et rouge seraient superposées durant cette période.

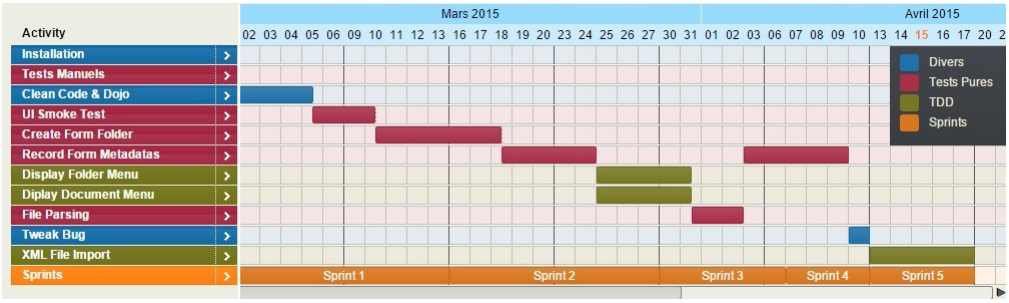
5.3 Plannification

Malgré une organisation SCRUM très précise, mon stage s'est déroulé durant une période de "rush", puisque la sortie du logiciel était imminante. Pour cela, mon tuteur me donnait les tâches a effectuer presque semaine par semaine car nous devions travailler par odre de priorité : les bugs et tests en premiers, puis viennent les nouvelles fonctionnalités.

Cependant, à l'aide d'un journal de bord que j'ai tenu, j'ai réalisé un diagramme de Gantt dont un extrait se trouve ci-après (cf. figure 5.5).

La figure suivante donne une idée de l'ensemble des tâches que j'ai réalisé durant mon stage. Ces tâches étaient variées et de plus en plus complexe au fur et à mesure du temps et m'ont permis de percevoir l'application sous plusieurs angles différents.

Figure 5.5 – Planning des tâches effectuées pendant le stage



En bleu figurent les différentes tâches d'installation, et de mise en place personnelle avant d'intégrer l'équipe de développement. En rouge sont représentés les tests que j'ai effectués. Les fonctionnalités que j'ai développé en TDD sont en vert. Enfin en orange sont représentés les différents sprints auxquels j'ai participé.

Conclusion

Ainsi, j'ai effectué mon stage au sein de DocuLibre. Lors de ce stage de douze semaines, j'ai pu mettre en pratique mes connaissances théoriques acquises durant ma formation, de plus, je me suis confronté aux difficultés réelles du monde du travail et du management d'équipes.

Après ma rapide intégration dans l'équipe, j'ai eu l'occasion de réaliser un ensemble de plusieurs missions variées et qui constituent la réussite de mon stage.

Chacune de ces missions, utiles au bon déroulement de l'activité de l'entreprise, se sont inscrites dans la stratégie de celle-ci : faciliter l'accès à l'information. Je garde du stage un excellent souvenir, il constitue désormais une expérience professionnelle valorisante mais surtout une expérience humaine plus d'enrichissante.

Je pense que cette expérience dans une entreprise canadienne m'a offert une bonne préparation à mon insertion professionnelle. En effectuant ce stage au Québec, j'ai appris de nouvelles méthodes de travailles différentes de la France. Cette expérience conforte mon désir d'exercer mon futur métier de « développeur logiciel » dans le domaine de l'informatique.

Enfin, je tiens à exprimer ma satisfaction d'avoir pu travailler dans de bonnes conditions matérielles et un environnement agréable, grâce à DocuLibre.

Références

— **Constellio** :
`http://www.constellio.com`

Sites du logiciel : site internet de l'entreprise par lequel on peut télécharger les codes sources du logiciel.

— **Ensemble de forums** [consultation pendant tout le projet] :
`http://www.commentcamarche.net/forum/`
`http://stackoverflow.com/`
`http://openclassrooms.com/forum/`

Sites associatifs : Ces forums m'ont servi pour résoudre un certain nombre de problèmes rencontrés. Les termes recherchés ont été par exemple : "click ne fonctionne pas Windows 8", "Sélénium click droit" ou encore "parsing fichier Open Office Custom Properties".

— **Slack** :
`http://slack.com`

Sites de participatif : site internet permettant de communiquer au sein de l'entreprise. Ce site très complet permet la communication privée ou par canal public ainsi que l'upload et le téléchargement de codes sources, fichiers, images, etc.

— **Martin, Robert Cecil, Clean code : a handbook of agile software craftsmanship, 2008** [consultation pendant tout le projet] :

Livre Technique : Le Clean Code est un ouvrage référençant les méthodes dites *propre* de codage.
En effet même un *mauvais* code peut fonctionner . Mais si le code n'est pas *propre*, il peut rendre la poursuite de son développement excessivement difficile. Chaque année , d'innombrables heures de travail et des ressources importantes sont perdues à cause de code mal écrit.
J'ai lu ce livre en entier et en version originale pendant le stage.

Résumé

Constellio est un logiciel open source dont le but est de permettre aux utilisateurs de gérer leur documents plus facilement. Le logiciel permet un traitement facile de l'information et met en place des fonctionnalités adaptées : moteur de recherche, gestion documentaire, archivage d'information. Le logiciel est développé en JAVA avec Vaadin. Il utilise la bibliothèque Sélénium et une base de données Apache SolR.

Mots Clés
Archivage d'information, Gestion documentaire, JAVA, Moteur de recherche, Open source, SolR

Constellio is an open source software which allows users to manage their documents easily. The software allows an easy processing of information and gives powerful functionality : search engine, document management, archiving documents . The software is developed in Java with Vaadin. It uses Selenium library and an Apache Solr database.

Key Words
Archiving documents, Document management, JAVA, Open source, Search Engine, SolR