IUT Nantes

Département Informatique

2015-2016

Rapport de stage

Réalisation d'un un outil d'exploitation automatique de documents Word

2ème année

11 avril au 17 juin 2016

Alex Medina

Encadré par: Thibault Castel
Mohamed amine AOUADHI

Remerciements

Je remercie Thibault Castel de m'avoir soutenu tout au long de ce stage.

Je remercie tous les professeurs de l'IUT de Nantes qui m'ont appris énormément de choses durant ces deux années de DUT informatique.

Je remercie également toute l'équipe de Lumiplan qui m'a accueilli chaleureusement.

Résumé

L'équipe du BEL* de Lumiplan Transport à Nantes a recourt à un gestionnaire de référentiel appelé Squash™, Ils utilisent ce logiciel pour rentrer principalement les exigences* et les cas de tests* de leurs projets, ces projets sont en relation directes avec leurs clients car ce sont les demandes des clients de Lumiplan. Une fois les exigences et cas de test rempli, ils peuvent générer des rapports Word avec chaque cas de test d'un projet. Cependant ces rapports ont une mise en page qui n'est pas présentable aux clients de Lumiplan. C'est ici que je dois intervenir en réalisant un outil d'exploitation automatique de documents Word. De plus quitte à utiliser Squash™, il fallait que je le mette à jour ou bien réaliser la documentation pour le faire.

Ce projet étant nouveau. Il fallait le mener du début à la fin, du cahier des charges, à l'intégration de l'application en passant par la documentation des différentes parties réalisés ou à faire par la suite du stage.

L'application devra prendre en entrer un fichier qui a été généré par Squash™, ainsi qu'un Template* Word pour définir la mise en page a adopté pour le document de sortie. Une fois le fichier source sélectionné, un arbre représentant les parties importante du document sera affiché sur une IHM*, l'utilisateur de l'application pourra sélectionner les éléments qu'il voudra garder dans le document final.

Le sujet de ce stage est très intéressant, car mené de bout en bout, mais il fallait être efficace, organisé et attentif pour le mener à bien dans son intégralité.

Abstract

Lumiplan of Transport in Nantes have employs a repository manager called SquashTM, They use this software to go mainly requirements and test cases of their projects, these projects are in direct relationship with their customers because it is the customer requests to Lumiplan. Once the requirements and test cases filled, they can generate Word reports for each test case of a project. However, these reports have a layout that is not presentable to customers of Lumiplan. It is here that I must intervene by performing automatic operating tool of Word documents. Moreover even use SquashTM, I need to update or complete the documentation to do so.

The application will have to enter a file that was generated by SquashTM and a Word Template to define the layout adopted for the output document. After selecting the source file, a tree representing the important parts of the document will be displayed on an HCI, the user can select the elements that he want to keep in the final document.

The subject of this course is very interesting, as led from start to finish, but I had to be efficient, organized and attentive to complete it in its entirety.

Tous les mots suivis d'une étoile sont expliqués dans le glossaire.

Table des matières

Introduction

Je suis actuellement en DUT informatique à Nantes, pendant ma formation, je dois réaliser un stage d'une durée de dix semaines dans une entreprise.

Ceci est donc mon rapport de stage qui est présenté comme cela :

La première partie du rapport sera consacrée à la présentation de Lumiplan, l'entreprise qui m'a accueilli pendant ces dix semaines de stage, le nom de mon maitre de stage était Thibault Castel.

La deuxième partie du rapport décrira le projet lui-même, en expliquant le contexte du projet ainsi que le besoin.

Ensuite j'expliquerai la conception de mes applications réalisées, c'est-à-dire comment je vais m'y prendre pour les concevoir.

Enfin je parlerai de la réalisation de celles-ci, c'est-à-dire les méthodes que j'ai utilisé pour mettre en œuvre mes applications..

Vous pouvez trouver tout le code réalisé ainsi que différents document sur mon GitHub

Lumiplan

1. Historique de Lumiplan

L'histoire de Lumiplan commence en 1972 par le lancement d'une idée : la création d'un plan lumineux pour informer les touristes. Stations balnéaires du littoral atlantique puis stations de ski, Lumiplan entame son développement.

En 1982, l'offre « Ski » s'étend aux Journaux électroniques d'information municipale.

En 1992, Lumiplan lance son activité Transport et participe aux réflexions de l'Union européenne sur les systèmes d'information destinés aux transports collectifs.

En 2002, Lumiplan intègre les solutions d'affichage « Full Colour » dans ses applications, et devient ainsi le premier acteur à déployer des écrans couleur extérieurs en France, notamment dans les enceintes sportives.

En 2007, Lumiplan remporte l'appel d'offre des Journaux électroniques d'information de la ville de Paris et consolide son offre Transport avec l'acquisition de l'activité Heurès, solution de graphicage et d'habillage de lignes de bus.

Lumiplan réalise 10 M€ de CA et emploie 55 collaborateurs.

En 2008, nouvelle acquisition : Lumiplan confirme son ancrage dans la ville avec le rachat de FA Technology, acteur important sur le segment de l'affichage dynamique urbain.

2009 marque un nouveau tournant : l'entreprise donne corps à ses ambitions internationales et crée la filiale Lumiplan Asia à Singapour.

2010, l'offre Transport s'accroît encore avec l'acquisition de Duhamel (Domène 38), pionnier des solutions de communication embarquées pour les transports publics.

Lumiplan emploie aujourd'hui 150 personnes, et dépasse les 30 M€ de CA.

2. Le groupe

Le groupe Lumiplan possède une présence à l'internationale dans près de 30 pays, certain membres de Lumiplan sont d'ailleurs amené à aller dans d'autre pays, en effet une personne venant du pôle Indien était présent à Nantes pendant toute la durée de mon stage. Étant donné qu'il ne parlait qu'anglais, j'ai trouvé cela très intéressant.

Le groupe est en constante croissance dans les domaines où ils travaillent.

Les solutions proposées par Lumiplan personnalisées et modulaires pour répondre parfaitement aux besoins des clients. Ils sont propriétaire des solutions développées par les équipes de R&D, cela permet une résolution rapide des problèmes que peuvent rencontrer les clients.

3. Les pôles

Le groupe est constitué de 4 pôles : ville, sport, montagne et transport, c'est dans ce dernier pôle situé à Nantes que j'ai réalisé mon stage.

Le pôle transport consiste à concevoir des solutions de communication pour le transport de voyageurs en bus, cars et tramway. Donner des informations aux voyageurs à bord et aux arrêts. Ils ont pour objectif de mieux informer les voyageurs, faciliter l'exploitation et optimiser l'offre de transport.

Ce sont les N°1 en France de l'information voyageur.

Ville : Plus de 1 500 communes équipées à ce jour dont Paris, Lille, Nantes, Dijon, Toulon, Bruxelles, Bruges... pour un parc total de plus de 2 200 journaux électroniques d'informations municipales en exploitation.

Transport embarqué : 6 000 véhicules équipés en systèmes de vidéo-protection, 30 000 girouettes en service, 1 200 véhicules équipés d'écrans TFT.

Sport : 2 000 m² d'écrans LED couleur qui équipent les enceintes sportives dont les stades de Lorient, Rennes, Annecy et le Palais omnisport de Montbéliard.

Montagne : L'équipement de 80% des stations de ski en France et aux USA, premier acteur avec les solutions à LEDs couleurs, brevet sur la gestion des flux de skieurs…

Présentation indépendante

1. Format OOXML

Avant de parler d'autres choses, j'aimerais faire une présentation du format Open Office XML pour faciliter la compréhension par la suite. Ce format a été créé par Microsoft et a été mis en place dans leur logiciel Microsoft Office 2007. Ce format prend place avec les fichiers dont l'extension est *.docx*, *.pptx* ou *.xlsx*, ce sont les extensions pour les fichiers Word, PowerPoint et Excel. Seul le format des documents Word va nous intéresser pour ce stage. Le format de ces fichiers sont représentés comme une archive, en effet, il est possible d'ouvrir un fichier Word avec un gestionnaire d'archive, à partir de là l'archive se décompose en multiple fichiers XML. Les propriétés ainsi que différentes informations sont contenus dans ces fichiers. Dans ce tas de fichiers, uniquement le fichier *document.xml* du dossier *word* va nous intéresser. Ce fichier contient tout le texte, les tableaux et références des hyperliens ou images.

2. Squash™

1. Présentation

Selon les développeurs,

> *SQUASH est un projet visant à développer un outillage open source mature et innovant pour contribuer à la structuration et à l'industrialisation des activités du test fonctionnel.*

Plus concrètement, il prend la forme d'un site web qui est déployé sur un serveur qui lui est propre, comme il s'agit d'un projet open source, nous pouvons réaliser toutes les modifications qui nous semblent nécessaire et par conséquent l'installer sur un serveur déjà existant. Il s'agit d'un projet réalisé par des français depuis 2011, il est toujours mis à jour à hauteur de deux mises à jour importantes par année. L'entreprise Lumiplan a décidé d'utiliser une base de donnés MySQL pour stocker les informations relative à Squash™. Ce site web installé sur un des serveurs de Lumiplan leurs permettent à l'équipe du BEL de rentrer des informations relative aux projets qu'ils ont avec leurs clients. Ils utilisent principalement Squash™ pour rentrer les exigences et les cas de test de leurs projets pour ensuite généré des rapports sur ces mêmes informations automatiquement.

La version que Lumiplan utilisait lors de mon arrivé dans l'entreprise était la version 1.12.3, elle utilisait un serveur Jetty* installé lui-même sur un serveur avec un système d'exploitation Debian*.

Seulement la version 1.13.3, la version à jour de Squash utilise elle un serveur TomCat*.

2. Le problème

Le problème est que les rapports Word générés par Squash™ ne sont pas présentables aux clients de Lumiplan sans faire d'importante retouche sur le document. De ce fait cette fonctionnalité n'est bonne à rien car l'équipe du BEL est obligée de refaire manuellement le rapport pour avoir une mise en forme présentable. Cela leur faire perdre énormément de temps car il peut s'agir de document de plus de 50 pages à refaire.

Présentation du sujet

Le but du projet est de donc de faciliter la création des rapports des cas de test pour permettre à l'équipe du BEL de gagner un temps considérable.

Pour travailler efficacement sur ce projet, j'allais avoir besoin d'une machine virtuelle, en effet le serveur sur lequel est déployé Squash™ est un serveur de production où est contenu un SVN*. Je ne pouvais donc pas agir à ma guise sur ce serveur.

1. Spécification des besoins

1. Besoins fonctionnels

Les besoins fonctionnels de l'application dans sa version finale sont les suivants :

- Le but de l'application est de modifier la mise en forme des documents générés par le serveur Squash, la mise en forme finale sera définie par un ou des Template.
- L'utilisateur pourra créer et modifier des Template, le texte, les images, la mise en forme et les styles présent dans ce Template seront gardés dans le document final.
- L'application finale devra avoir une IHM permettant à l'utilisateur de réaliser toutes les actions nécessaires, aucune action ne devra être réalisée en ligne de commande, de plus l'application devra pouvoir être exécutable sous Windows pour avoir une meilleure portabilité.
- Dans un second temps, l'application pourra prendre en entrée un ou des documents de différents types (docx, txt, …). Cette partie ne sera surement pas réalisable dans le temps

impartie du stage, mais fera surement office d'amélioration réalisée par l'équipe du BEL ou lors d'un autre stage.

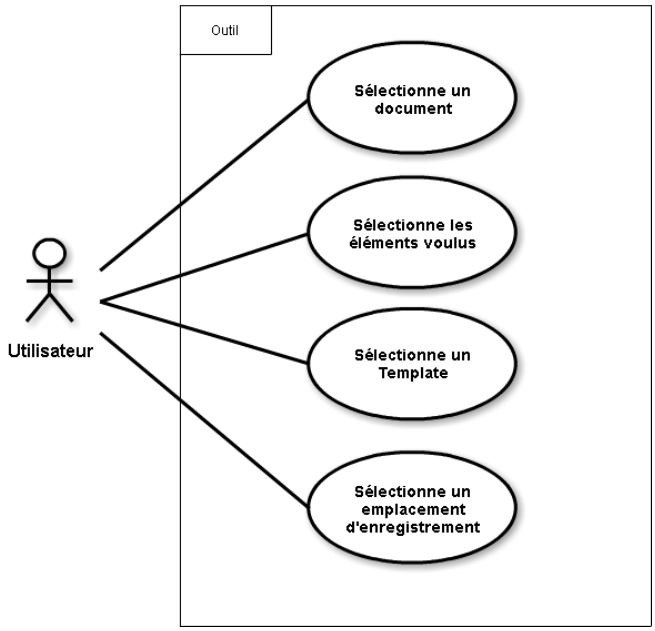
- Suite à l'ouverture du document généré par Squash, un arbre représentant son contenu sera affiché dans l'IHM.
- Les éléments de l'arbre seront regroupés selon plusieurs critères, tout d'abord, la plupart des éléments de l'IHM seront cochables, elle aura donc la forme d'une check-list. Les éléments communs aux fiches de test seront regroupés dans un seul onglet pour ne pas surcharger l'IHM. Les fiches de test pourront avoir un élément père dans l'arbre, le père, s'il existe, sera le nom du dossier dans lequel la fiche est contenue.
- Si les éléments sont décochés, ils ne seront pas présents dans le document final.
- Dans l'arbre, sera ajouté à la demande des éléments cochables, qui auront comme forme finale des booléen.
- Une fois le choix des éléments terminé, l'IHM demandera à l'utilisateur de choisir un Template créé auparavant.
- L'IHM demandera par la suite, de choisir l'emplacement de l'enregistrement du document final.
- Les éléments sélectionnés auparavant dans l'IHM seront écrits dans le document final selon la mise en forme et les styles du Template.
- L'application se fermera ensuite automatiquement lorsque le document sera généré.

2. Besoins non fonctionnels

Les besoins non fonctionnels de l'application dans sa version finale sont les suivants :

- L'application devra construire l'arbre en moins de 3 secondes après l'ouverture du document généré par Squash.
- Elle devra générer le document final en moins de 3 secondes après le choix d'enregistrement du fichier.
- Le langage utilisé pour réaliser cette application sera le Python. Ce choix est dû au fait que les membres du BEL maitrisent le Python.
- L'application ne doit pas consommer suffisamment de ressources pour surcharger le PC.
- Elle devra pour fonctionner avec n'importe quel document généré par Squash sans générer d'erreur.
- Les scripts Python devront respecter la norme PEP8.
- L'IHM devra être intuitive et ne sera pas surchargée d'éléments pour une meilleure visibilité. Elle sera donc facile d'utilisation.

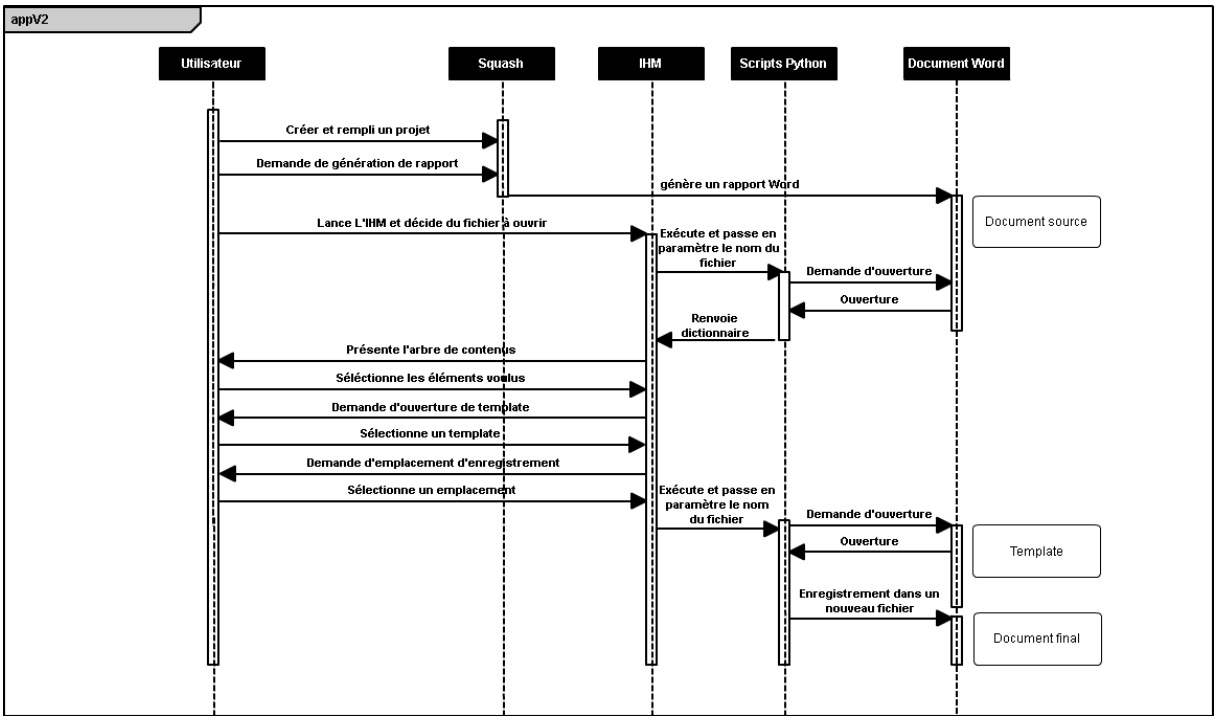
2. Diagramme use case



Dans sa version finale, l'utilisateur aura seulement ces actions à réaliser pour utiliser correctement l'application. Il pourra également créer lui-même ses Template. Cependant ces actions devront être faites dans un certain ordre, l'utilisateur ne pourra pas se tromper car il sera guidé par l'IHM.

3. Diagramme de séquence

Voici comment l'application devra fonctionner dans sa version finale.



Conception

1. Déroulement du stage

1. Outil à disposition

Un ordinateur, un clavier, une souris, un écran, ainsi qu'un câble Ethernet m'ont été fournis lors de mon arrivé, j'ai dû installer mon poste de travail. J'ai ensuite récupérer les informations de connexions de mon poste ainsi qu'à une boite mail de l'entreprise par la suite.

Sur mon ordinateur, j'ai reçu les droits d'administrateur pour pouvoir installer tous les logiciels qui allaient m'être utile (Python, Sublime Text comme éditeur de texte, VMWare pour une machine virtuelle), dans la semaine de mon arrivé, une image Debian m'a été fournis par Thibault Castel.

2. Encadrement

Durant ce stage, j'ai été encadré par Thibault Castel, le responsable du BEL avec qui j'avais passé mon entretien. Le BEL est divisé en deux bureau, celui des développeurs et celui des responsables. C'est dans ce dernier que j'ai été placé, cela m'a permis d'être au plus près de mon maitre de stage et de ses collègues qui m'ont également aidé lorsque j'en avais besoin. L'ambiance au sein du bureau était très agréable et j'ai été invité à divers événements non professionnel (repas).

2. Cahier des charges

J'ai dû rédiger un cahier des charges durant les premières semaines de mon stage. Il m'a permis de bien comprendre les besoins, et de définir les fonctionnalités attendues. Il m'a aussi servi de guide tout au long du développement.

Vous pouvez retrouver ce cahier des charges sur mon GitHub.

3. Choix de réalisation

Pour réaliser ce travail, j'avais plusieurs possibilités de conceptions.

A cause d'un manque de connaissance en début de stage, j'ai d'abord pensé à modifier directement le Template de génération des rapports de Squash^{TM} pour avoir directement une mise en forme convenable, cependant il y aura des retouches à réaliser par la suite car la génération ne pourra pas être parfaite. C'est pourquoi j'ai ensuite pensé à modifier directement le fichier *document.xml* du document pour résoudre les problèmes de mise en forme qu'il pouvait rester.

Dans un second temps j'ai découvert de nouvelles librairies qui allaient pouvoir me faciliter la tâche dans la création du nouveau document. Pour réaliser l'application selon ces librairies, j'allais devoir récupérer tout le texte du document et le sauvegarder dans un dictionnaire. Une IHM serait également présente pour permettre à l'utilisateur des options de création du fichier final.

J'ai donc réalisé ces deux applications, la première était pour permettre à l'équipe du BEL de travailler plus facilement dans un premier temps, en attendant la deuxième version de l'application plus élaborée.

4. Étude technique

Pour me permettre de réaliser correctement mon application, j'ai recherché des moyens de modifier des documents Word, j'ai donc pu trouver des librairies me permettant de réaliser le travail que j'allais devoir effectuer

- lxml est une librairie permettant d'interagir avec les fichiers XML.
- zipfile permet de créer une archive et de récupérer des fichiers d'une archive.
- shutil permet de copier, de déplacer ou de supprimer des fichiers ou dossiers.
- docx permet de créer ou de récupérations les informations d'un document Word.
- docxtpl permet d'écrire des informations dans un document Word selon un Template.
- Tix et ses modules complémentaires permettent de créer des IHM.
- py2exe permet de créer un exécutable à partir de scripts Python.

Je vais m'intéresser d'avantage aux librairies *docx* et *docxtpl.* Tout d'abord *docx*, cette librairie particulièrement intéressante pour moi va me permettre de pouvoir récupérer les informations du document selon des paragraphes ou des selon les tableaux.

Dans un document Word, chaque paragraphe est séparé par un saut de ligne. Dans chaque paragraphe, il se trouve des *runs,* chaque run correspond à un morceau de texte qui a un style propre, par exemple dans le paragraphe :

> Un texte avec du **gras** et de *l'italique.*

Il va y avoir 4 runs ; la première comprenant : « Un texte avec du », la seconde « gras », la troisième « et de » pour finir par « l'italique ». Les runs possèdent donc des propriétés pour savoir le style du texte.

Cette librairie peut également écrire dans des fichiers Word, mais cette fonctionnalité ne va pas nous intéresser car on ne peut ajouter qu'en début de document.

Pour écrire dans un document Word, je vais utiliser la librairie *docxtpl* qui permet d'écrire des informations donnés dans un endroit donné. La librairie utilise un dictionnaire pour fonctionner, dans notre Template nous allons écrire des tags entre des balises spécifiques qui vont correspondre aux clés de notre dictionnaire.

Les informations qui vont ensuite être écrite dans ces tags seront les valeurs de nos clés correspondant au tag.

Je vous invite à regarder la documentation pour de plus amples informations.

5. La première application

La première application que j'ai faite était simpliste, elle a permis aux membres de l'équipe du BEL de pouvoir rédiger des rapports sans option de modification lors de l'exécution de l'application. Pour réaliser cette application, j'ai d'abord dû étudier les rapports de Squash^{TM} qui sont comme suit :

1. Tests Fonctionnels > Serveur > 1 Mise en place IHM

1.1. Fiche 1 - Accès à l'IHM

Créé le :	31 juil. 2015 16:36 (par christine.salotti)
Modifié le :	14 août 2015 18:22 (par christine.salotti)
ID:	530
Nature :	Non définie
Type:	Non défini
Statut :	Approuvé
Importance:	Moyenne
Jalons:	

DESCRIPTION

Valider l'acces à l'interface du SIV.

Pré-requis :

Exigences couvertes :

- Roissy Pôle
  - - P015 - Mise en place d'une interface web de supervision des panneaux (MINOR)
  - - P016 - Gestion de 3 profils (MINOR)

ETAPE 1 :

Action	**Accéder à l'interface web :** Saisir l'adresse : http://192.168.1.93/siv
Résultat attendu :	La console SIV est accessible en HTTP

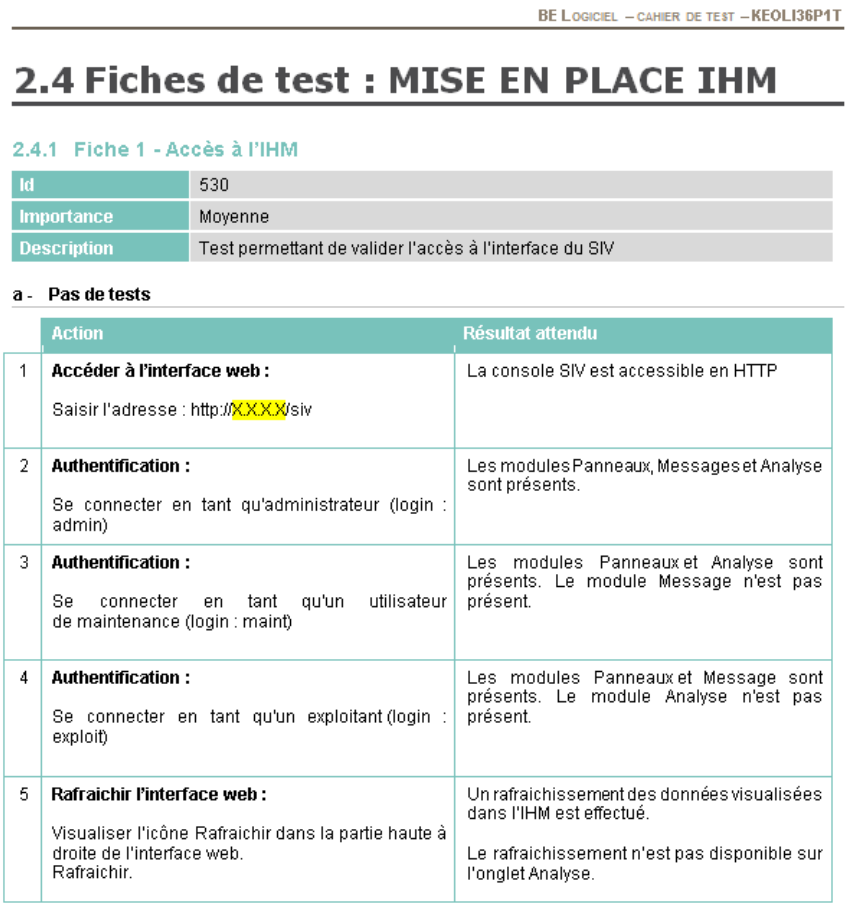
- Pièces jointes : Non
- Exigence : Aucune

On peut voir ici différents éléments importants d'une fiche de test:

- Le dossier (encadré en haut)
- Le nom de la fiche
- Créé le
- Modifié le
- ID
- Nature
- Type
- Statut
- Importance
- Jalon
- Une description de la fiche
- Ses exigences
- Puis ses cas de test défini par étape comprenant une action et un résultat attendu, les étapes peuvent être nombreuses, j'ai décidé de n'en montrer qu'une seule.

Ensuite, la fiche suivante va être affichée avec les mêmes éléments listés ci-dessus. Selon le nombre de fiches, le rapport peut vite atteindre un nombre important de page. Ce rapport faisait plus de 50 pages après génération.

Cependant les rapports faits par Lumiplan ont une mise en forme comme celle-ci :



BE LOGICIEL – CAHIER DE TEST – KEOLI36P1T

2.4 Fiches de test : MISE EN PLACE IHM

2.4.1 Fiche 1 - Accès à l'IHM

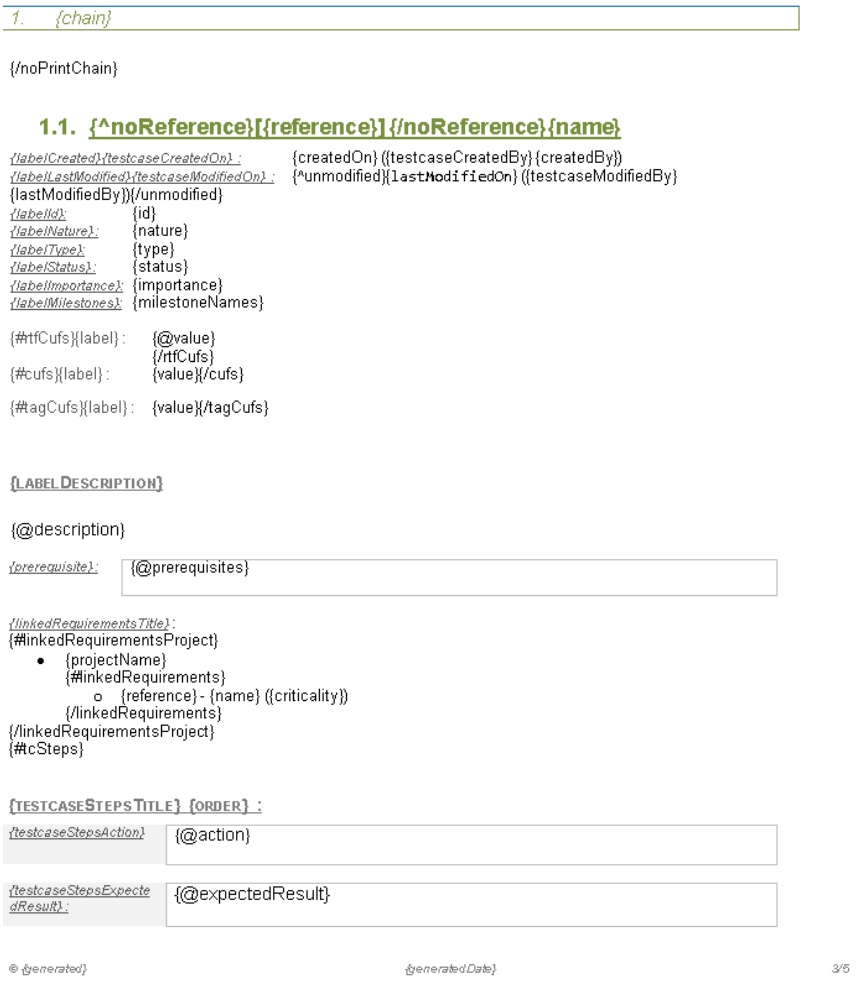
Id	530
Importance	Moyenne
Description	Test permettant de valider l'accès à l'interface du SIV

a - Pas de tests

	Action	Résultat attendu
1	**Accéder à l'interface web :** Saisir l'adresse : http://X.X.X.X/siv	La console SIV est accessible en HTTP
2	**Authentification :** Se connecter en tant qu'administrateur (login : admin)	Les modules Panneaux, Messages et Analyse sont présents.
3	**Authentification :** Se connecter en tant qu'un utilisateur de maintenance (login : maint)	Les modules Panneaux et Analyse sont présents. Le module Message n'est pas présent.
4	**Authentification :** Se connecter en tant qu'un exploitant (login : exploit)	Les modules Panneaux et Message sont présents. Le module Analyse n'est pas présent.
5	**Rafraichir l'interface web :** Visualiser l'icône Rafraichir dans la partie haute à droite de l'interface web. Rafraichir.	Un rafraichissement des données visualisées dans l'IHM est effectué. Le rafraichissement n'est pas disponible sur l'onglet Analyse.

On constate que certain éléments ont été supprimés, qu'ils ont été placés dans des tableaux pour faciliter la lisibilité, de plus cette mise en forme permet un gain de place considérable, en effet, pour le même document que précédemment avec cette mise en forme, le rapport ne fait que 24 pages.

On parcourant les fichiers de configurations de Squash™, j'ai pu trouver le Template qui permet de générer les documents Word, le document est le suivant :


1. {chain}

{/noPrintChain}

1.1. {^noReference}[{reference}] {/noReference}{name}

{labelCreated}/{testcaseCreatedOn} : {createdOn} ({testcaseCreatedBy}{createdBy})
{labelLastModified}/{testcaseModifiedOn} : {^unmodified}{lastModifiedOn} ({testcaseModifiedBy}{lastModifiedBy}){/unmodified}
{labelId}: {id}
{labelNature}: {nature}
{labelType}: {type}
{labelStatus}: {status}
{labelImportance}: {importance}
{labelMilestones}: {milestoneNames}

{#rtfCufs}{label} : {@value}{/rtfCufs}
{#cufs}{label} : {value}{/cufs}

{#tagCufs}{label} : {value}{/tagCufs}

{LABEL DESCRIPTION}

{@description}

{prerequisite}: {@prerequisites}

{linkedRequirementsTitle} :
{#linkedRequirementsProject}
- {projectName}
  {#linkedRequirements}
  - {reference} - {name} ({criticality})
  {/linkedRequirements}

{/linkedRequirementsProject}
{#tcSteps}

{TESTCASESTEPSTITLE} {ORDER} :

{testcaseStepsAction}	{@action}
{testcaseStepsExpectedResult} :	{@expectedResult}

© {generated} {generatedDate} 3/5

On y retrouve tous les éléments mentionnés plus haut.

Les éléments sont placés aux bons endroits selon les tags utilisés, on constate que les tags avec des # représente des boucles, par exemple : *{#tcSteps}* va faire une boucle sur chaque cas de test de la fiche actuelle, cela va permettre d'afficher les cas de test les uns à la suite des autres tout en gardant la mise en forme défini dans ce Template.

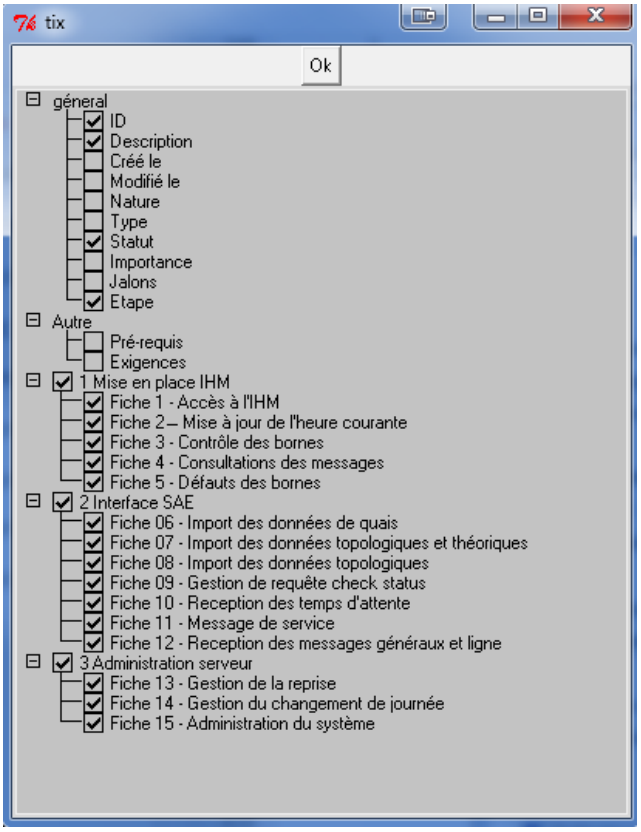
En modifiant ce Template, je peux donc modifier la mise en forme des rapports générés par Squash, cela est assez complexe car à la moindre erreur dans le document, le document ne se génère pas et il n'y a aucun moyen de savoir pourquoi cela ne fonctionne pas. Cependant, je savais que je ne pourrais pas obtenir de résultat parfait juste en modifiant ce Template, c'est pourquoi des scripts Python m'allaient être nécessaires pour finaliser la mise en forme du document.

Ces scripts allaient modifier directement dans le fichier *document.xml* les parties qui n'allaient pas me servir.

Je vais expliquer en détails comment j'ai réalisé cette application dans la partie « Réalisation ».

6. La seconde application

La deuxième application réalisée est plus complexe, elle dispose d'une IHM simpliste guidant l'utilisateur à travers l'application. Cette application consiste à utiliser le Template de base de Squash™ pour générer les documents, cela veut donc dire que le document en entré de l'application sera non modifié de base, une fois le document sélectionné, un arbre sous la forme d'une checkList apparaitra demandant à l'utilisateur de sélectionner les éléments qu'il veut garder. L'IHM aura donc cette forme :

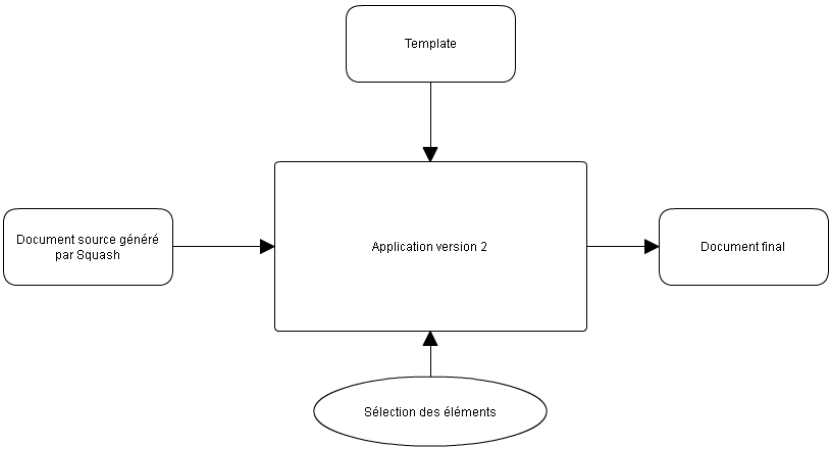


On peut voir que les éléments communs à chaque fiche sont représentés dans un onglet général.

Dans un autre onglet, les prérequis et exigences sont affichés pour faciliter la mise en page lors de la création des Template, en effet l'équipe de Lumiplan préfère avoir ces éléments à part pour qu'ils fassent partis d'un autre tableau.

Puis chaque fiche est affichée. Dans l'arbre, les fiches ont un père qui correspond au dossier où est stockée cette fiche dans Squash.

Une fois les éléments sélectionnés et le bouton « Ok » appuyé, l'IHM va demander à l'utilisateur de choisir un Template pour la mise en forme, elle va ensuite demander de choisir un emplacement pour enregistrer le document final.



Le but est de pouvoir avoir un maximum d'options lors de la création du document final, via l'IHM et via le Template qui doit être modifiable par l'utilisateur quand il le souhaite. Pour Lumiplan, l'objectif est de créer un Template par équipe : panneau, serveur, électronique, affaire.

Je vais expliquer en détails comment j'ai réalisé cette application dans la partie « Réalisation ».

Réalisation

1. La première application

Cette application a été rapide à coder mais assez complexe, comme j'ai dû modifier le document XML directement, il me fallait une parfaite connaissance de celui-ci, cependant les fichiers générés par Squash ont un fichier XML qui n'est pas propre du tout. Les *run* des paragraphes sont découpés n'importe comment, parfois, il y a une *run* par lettre alors que celles-ci ont chacune le même style. De plus je ne pouvais pas utiliser la librairie *lxml* pour modifier le fichier XML de ce fait, je ne savais pas de base quelle partie j'allais devoir supprimer, partie que je vais définir plus tard.

J'ai donc commencé cette application par modifier le Template de Squash. Dans ce Template, j'ai surtout supprimé les éléments qui n'étaient pas dans le rapport fait par Lumiplan. J'ai également refait la mise en forme du document, finalement, le Template de Squash ressemble à ceci :

{#testCases}{^AFolder}{^noPrintChain}{/noPrintChain}

1. {^noReference}[{reference}] {/noReference}{name}

{labelId}:	{id}
{labelImportance}:	{importance}
{labelDescription}	{@description}
{prerequisite}	{@prerequisites}

Statut :

CONFORME	NON CONFORME	N/A	NON EXECUTE
☐	☐	☐	☐

a - Pas de tests

{#tcSteps}

	Action	Résultat attendu
{order}	{@action}	{@expectedResult}

{/tcSteps}

{/AFolder}{/testCases}
{/projects}

On y retrouve la mise en via des tableaux comme sur le document original de Lumiplan.

Créer ce document, malgré les apparences a été complexe, en effet, à la moindre erreur le document refusait de se généré, cela a été un des plus gros problèmes que j'ai dû surmonter pour réaliser cette application. Par exemple, il faut obligatoirement qu'il y ait une ligne vide à la fin des tableaux sinon le document est faux, et le seul moyen de le savoir était de le deviner. Il n'existe aucune documentation pour cela. J'ai dû procéder petit à petit pour construire ce document. De plus pour le tester, je devais remplacer ce fichier dans une archive de Squash et redémarrer le serveur sur ma machine virtuelle. En moyenne le serveur mettait 3 minutes pour se relancer. Il fallait donc être prudent lors de la rédaction de ce document.

Le document généré par ce Template ressemblait donc à celui-ci :

13.Fiche 13 - Gestion de la reprise

ID:	534
Importance:	Haute
Description	Test permettant de contrôler que le SIV est capable de reprendre l'envoi de trame à une borne si celui-ci a été interrompu.
Pré-requis	

Statut :

CONFORME	NON CONFORME	N/A	NON EXECUTE
☐	☐	☐	☐

a - Pas de tests

	Action	Résultat attendu
1	**Déconnexion de la borne :** Couper l'alimentation électrique de la borne	Dans l'IHM du SIV la borne apparaît avec un défaut « Communication » La mise à jour des défauts est faite cycliquement toutes les 2 minutes.
2	**Réception d'un message général :** Creer dans le SAE un message général à destination de la borne de test Attendre quelques secondes.	
3	**Connexion de la borne :**	Dans l'IHM du SIV la borne n'apparaît plus avec un défaut « Communication ».

Ce document commence à ressembler à celui que propose Lumiplan, le tableau « statut » a été rajouté suite à la demande d'un des membres du BEL, le problème est qu'il y a des sauts de ligne entre chaque ligne du tableau et une ligne vide à la fin de chaque cellule du tableau. Les scripts Python vont mettre utile pour supprimer ces parties ainsi qu'à définir une police et une taille d'écriture par défaut.

Pour pouvoir modifier ce document via le fichier *document.xml*, j'ai d'abord dû créer une copie du fichier, tout en renommant l'extension en *.zip* pour pouvoir utiliser la librairie *zipfile* dessus.
Une fois le fichier zip créé, j'ai récupérer grâce la librairie *zipfile* le fichier *document.xml*, j'ai ensuite dû refaire l'arbre XML du fichier car celui se trouvait sur une seule ligne, je devais l'avoir sur une ligne par balise pour pouvoir le lire correcte visuellement ou via des scripts, j'ai donc utilisé la librairie *lxml.etree*, voici la méthode qui permet de le faire :

```python
def xml(file, fileName):
    """Fonction qui met en forme le fichier xml et
    le déplace dans le dossier ./../xml/
    en le renommant selon son nom exact suivi de '_document'.
    """

    #  Fermeture automatique à la fin du traitement.
    with open("./../xml/" + fileName + "_document.xml", 'w') as f:
        #  Créer l'arbre xml et l'écrit dans le fichier
        f.write(etree.tostring(etree.parse(open(file)), pretty_print=True))

    os.remove(file)  # Supprime document.xml du dossier word

```

Après ça, j'ai pu analyser correctement le fichier *document.xml*.
Après plusieurs tentatives fortuites, j'ai réussi à correctement faire le travail que je voulais.
Pour les sauts de ligne, j'ai remarqué qu'ils prenaient forme de paragraphes sans *run* à l'intérieur. J'ai donc enregistré dans un buffer chaque lignes après le début d'un paragraphe, si je rencontrais une *run*, j'écrivais le contenu de mon buffer, si je rencontrais la fin de mon paragraphe, je supprimais mon buffer.
Pour la police d'écriture, j'ajoutais après chaque *run* des propriétés pour la police d'écriture ainsi que sa taille.

Ces scripts m'ont permis d'avoir un document sous cette forme :

13.Fiche 13 - Gestion de la reprise

ID:	534
Importance:	Haute
Description	Test permettant de contrôler que le SIV est capable de reprendre l'envoi de trame à une borne si celui-ci a été interrompu.
Pré-requis	

Statut :

Conforme	Non conforme	N/A	Non executé
☐	☐	☐	☐

a - Pas de tests

	Action	Résultat attendu
1	**Déconnexion de la borne :** Couper l'alimentation électrique de la borne	Dans l'IHM du SIV la borne apparaît avec un défaut « Communication » La mise à jour des défauts est faite cycliquement toutes les 2 minutes.
2	**Réception d'un message général :** Creer dans le SAE un message général à destination de la borne de test Attendre quelques secondes.	.
3	**Connexion de la borne :** Rallumer la borne.	Dans l'IHM du SIV la borne n'apparaît plus avec un défaut « Communication ». Le message général envoyé au pas 2 apparait sur la borne.

Le résultat est assez satisfaisant, cependant ils y a d'énorme contrainte :

- Il s'agit d'une modification interne du Template de Squash
- La modification du Template est complexe
- Un seul Template possible
- Aucune option de réalisation
- Un document final unique
- Des erreurs peuvent survenir lors de l'ouverture du document final si une cellule n'a pas été remplie dans Squash

De plus il faut faire deux modifications à l'intérieur du document, à savoir de changer la pagination des tableaux pour éviter qu'ils ne se coupent en milieu de page, il faut aussi désactiver les espaces entre les paragraphes du même style, ces options sont toutes dans l'onglet « Paragraphe » de Word.

Cette application a donc pas mal de défauts cependant, elle aura permis dans un premier temps de rédiger des rapports plus facilement pour l'équipe du BEL.

2. La seconde application

1. Les scripts

Cette application a été plus longue et complexe à réaliser mais elle était aussi nettement plus intéressante. Par rapport à la première application faite, je suis pour ainsi dire reparti de zéro. Je suis donc reparti avec le document de base. Tout d'abord lors du lancement de l'application, une IHM s'ouvre en demandant d'ouvrir le document généré par Squash.

Une fois ouvert, j'envoie ce document dans un autre script qui va récupérer les données du fichier. J'ai utilisé la librairie *docx* pour lire séparément les paragraphes et les tableaux, étant donné que la structure du document est toujours la même, j'ai pu établir des contrôles pour savoir à quelle partie du texte j'en étais dans ma lecture. A partir de là, j'ai établi un dictionnaire assez conséquent. Voici la forme finale de mon dictionnaire :

```
dictionnaire = {

    'general': [Liste des éléments sélectionnés dans l'ihm qui ont pour père 'general'. List of String],

    'num': [Liste de 1 jusqu'au nombre de fiches, List of Integer],

    'Other': [
        {
            'Pre-requis': RichText(String), 'Exigences': RichText(String)
        }
        ...
    List of Dict]

    'is_Etape': Boolean

    'is_Statut': Boolean

    'Fiches': [
        {
            'Titre': String, 'Parent': String, 'ID': Integer, 'Description': String, 'Cree': String, 'Modifie': String,
            'Nature': String, 'Type': String, 'Statut': String, 'Importance': String, 'Jalons': String,
            'is_children': Boolean, 'other_father': Boolean
            'Etapes': [
                {
                    'Numero': RichText(Integer), 'Action': RichText(String), 'Resultat': RichText(String)
                },
                ...
            List of Dict],
        },
        ...
    List of Dict]
}
```

Ps : Nous verrons plus tard ce qu'est le « RichText »

La première partie de mon script correspond à la création de « Fiches », je cherche à récupérer toutes les informations intéressantes en les catégorisant via les clefs de mes dictionnaires dans « Fiches ». Pour récupérer ces informations, j'ai défini une liste de clefs qui sont présentes dans le document d'origine

```
36      # Liste des éléments à récupérer dans le texte, drop 'Statut'
37      element = ('Modifié', 'ID', 'Nature', 'Type',
38                 'Importance', 'Jalons',
39                 'ID:', 'Type:', 'Importance:', 'Jalons:')
```

Statut n'est pas recherché car il existe le Template.

Tous les éléments ne sont pas présent car pour certain, il y a des méthodes particulières pour récupérer les valeurs associés. Par exemple pour récupérer la description ainsi que les prérequis, il faut faire :

```
98
99          # Lorsqu'il y a Description dans le paragraphe.
100         if 'Description' in elem.text:
101             i = 2
102
103             # On avance jusqu'à ce qu'il n'y ai plus de texte
104             while para[para.index(elem) + i].text:
105                 i += 1
106
107             # On enregistre le texte qui nous intéresse.
108             dico['Description'] = "\n".join([para[para.index(elem) + i].text for i in range(2, i+1)])
109
110             # On sait que les pré-requis sont après la description.
111             # On va donc chercher dans les tables les pré-requis
112             cell = tt.next()
113             dico['Pre-requis'] = cell.row_cells(0)[1].text
114
```

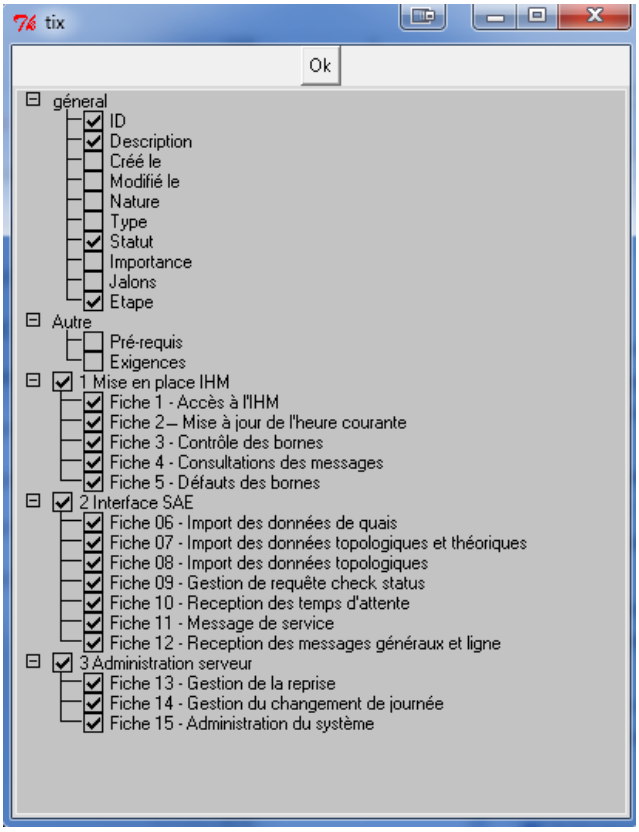
On peut voir que j'utilise les listes par compréhension de Python pour améliorer les performances de mon script.

Alors que pour récupérer les éléments communs il suffit de :

```
157         # Pour les autres éléments
158         if any(str in element for str in(elem.text.split())):
159
160             # On récupère le 1er mot de l'élément
161             key = elem.text.split()[0]
162
163             # On va remplacer les caractères spéciaux.
164             for src, dest in replacements.iteritems():
165                 key = key.split()[0].replace(src, dest)
166
167             # On va ajouter à notre dictionnaire, tout ce qu'il y a après le :
168             tmp = elem.text.split(':')
169             dico[key] = ':'.join(tmp[1:])
170
```

J'ai décidé d'enlever les accents que j'ai définis dans les clefs de mon dictionnaire pour éviter certains problèmes d'encodage.

Une fois tout le contenu de mes fiches représenté dans mon dictionnaire, j'envoie ce dictionnaire à mon IHM, elle va commencer par afficher les éléments communs aux fiches que j'ai écrits manuellement, c'est-à-dire les éléments dans l'onglet général et autre. Ensuite, elle va trier les informations qui nous intéresse, c'est-à-dire les titres des fiches et le nom de leur parent. Au final, cela va donner une IHM comme celle-là :

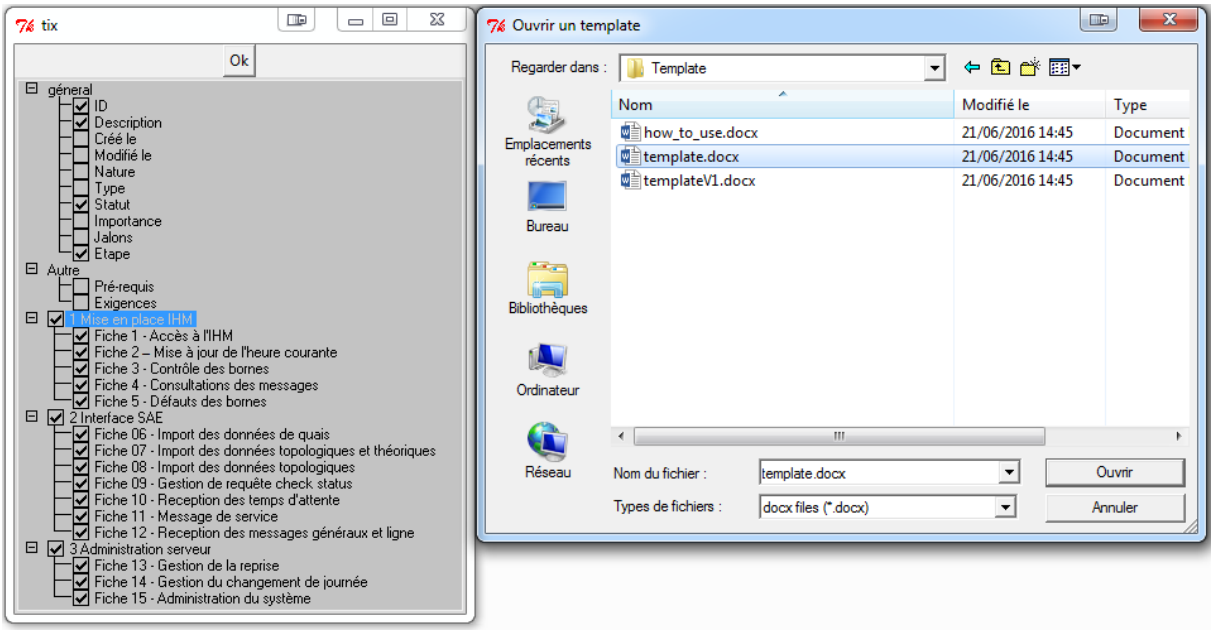


Cet arbre apparait une à deux secondes après le choix du document source.

Les éléments sont cochés comme cela de base par choix de l'équipe du BEL de Lumiplan.

S'il on ne veut faire aucune modification, il suffit d'appuyer sur le bouton « Ok »

L'application va ensuite demandée d'ouvrir un Template et de choisir l'emplacement de destination.



Une fois cela terminé, l'IHM va envoyer une liste des éléments qui sont cochés à un autre script Python. Celui-ci va écrire les derniers éléments de notre dictionnaire, tout en supprimant les fiches non sélectionnés pour éviter de surcharger le dictionnaire.

```
219
220     # Pour chaque fiches, on va supprimer les éléments indésirables
221     # La liste est inversée pour éviter les problèmes de supression
222     for fiche in reversed(d2['Fiches']):
223         # On récupère l'index de la fiche courante.
224         index = d2['Fiches'].index(fiche)
225
226         if fiche['Parent']:
227             if fiche['Parent'] + '.' + fiche['Titre'] not in active:
228                 d2['Fiches'].pop(index)
229         else:
230             # Si la fiche n'est pas un élément souhaité, on la suprimme
231             if fiche['Titre'] not in active:
232                 d2['Fiches'].pop(index)
233
234             # On passe à la fiche suivante
235             continue
236
237         if 'general.Etape' in active:
238             d2['is_Etape'] = True
239         else:
240             d2['is_Etape'] = False
241
242         if 'general.Statut' in active:
243             d2['is_Statut'] = True
244         else:
245             d2['is_Statut'] = False
246
247     # On ajoute dans le dictionnaire, une liste de 1 au nombre de fiches.
248     d2['num'] = [i + 1 for i in range(len(d2['Fiches']))]
249
```

Encore une fois à la ligne 248 j'utilise une liste par compréhension de Python.

Le dictionnaire est pratiquement terminé, avant d'écrire dans notre Template, il faut avant passer certaine partie en RichText,

Pour information, la librairie *docxtpl* qui permet d'écrire dans notre Template utilise la librairie *docx*, et cette librairie va écrire dans le fichier XML, tout n'est donc pas permis, en effet si l'on décide d'écrire des chevrons dans un fichier XML, cela va causer pas mal de soucis car les il ne les différenciera pas des balises, le RichText permet d'éviter cela en plus de traduire les « \n » par des « </br> » dans le fichier pour permettre le saut de ligne.

Cependant chaque partie ne nécessite pas ce texte riche, seulement les prérequis ainsi que les exigences sinon ils auraient un affichage qui n'est pas propre.

```
    for fiche in dictionnaire['Fiches']:
        for key1, value1 in fiche.iteritems():
            if(isinstance(value1, basestring) and
               ('Exigences' in key1 or 'Pre-requis' in key1)):

                value1 = value1.replace('\t', '')
                while value1.endswith('\n'):
                    value1 = value1[:-2]
                while value1.startswith('\n'):
                    value1 = value1[1:]
                fiche[key1] = RichText(value1)

            elif isinstance(value1, list):
                for elem in value1:
                    for key2, value2 in elem.iteritems():
                        elem[key2] = RichText(value2)
```

Il faut ensuite demander d'écrire dans notre Template.

2. Les Template

Avant d'écrire dans notre Template, il faut d'abord le créer, pour cela il faut respecter certaines règles établi par la librairie pour ne pas être redondant et être performant dans sa création.

Je vous invite à regarder la documentation de *docxtpl* ou de vous reporter au fichier *how_to_use.docx* disponible sur mon *GitHub*.

Voici donc le Template que j'ai créé dans sa version finale :

DÉROULEMENT DES TESTS

{%r for fiche in Fiches%}
{%p if fiche.is_children %}{%p if fiche.other_father %}

{{fiche.Parent}}

{%p endif %}{%p endif %}

1. {{fiche.Titre}}

{%tr for gen in general %}	
{{gen}}	{{fiche[gen] }}
{%tr endfor %}	

{%tr for autre in Other %}	
{{autre}}	{{r fiche[autre] }}
{%tr endfor %}	

{%p if is_Statut%}

Statut :

Conforme	Non conforme	N/A	Non exécuté
☐	☐	☐	☐

{%p endif %}
{%p if is_Etape%}

	Action	Résultat attendu
{%tr for etape in fiche.Etapes%}		
{{r etape.Numero}}	{{r etape.Action}}	{{r etape.Resultat}}
{%tr endfor %}		

{%p endif %}
{%r endfor %}

On y retrouve les clefs qui sont présente dans mon dictionnaire.

La mise en forme présente sur ce Template restera sur le document final, ainsi tout texte écrit.

Encore une fois je vous invite à vous référer à la documentation sur mon GitHub.

3. Le résultat

Une fois toutes les instructions de l'IHM suivi, votre document final est créé là où vous l'avez mentionné. Il ne vous reste plus qu'à l'ouvrir.

DÉROULEMENT DES TESTS

1 Mise en place IHM

1. Fiche 1 - Accès à l'IHM

ID	530
Description	Valider l'accès à l'interface du SIV.

Statut :

Conforme	Non conforme	N/A	Non exécuté
☐	☐	☐	☐

	Action	Résultat attendu
1	Accéder à l'interface web : Saisir l'adresse : http://192.168.1.93/siv	La console SIV est accessible en HTTP
2	Authentification : Se connecter en tant qu'administrateur (login : admin)	Les modules Panneaux, Messages et Analyse sont présents.
3	Authentification : Se connecter en tant qu'un utilisateur de maintenance (login : maint)	Les modules Panneaux et Analyse sont présents. Le module Message n'est pas présent.
4	Authentification : Se connecter en tant qu'un exploitant (login : exploit)	Les modules Panneaux et Message sont présents. Le module Analyse n'est pas présent.
5	Rafraichir l'interface web : Visualiser l'icône Rafraichir dans la partie haute à droite de l'interface web. Rafraichir.	Un rafraichissement des données visualisées dans l'IHM est effectué. Le rafraichissement n'est pas disponible sur l'onglet Analyse.

Le résultat est identique à l'original avec le tableau du statut en plus. Cependant, le style du gras, et de l'italique ou des listes à puces n'est pas conservé dans ce document car je n'ai pas eu le temps de l'implémenter, j'ai cependant fournis une documentation avec les conseils pour les réaliser sur mon GitHub.

Par rapport à l'application précédente, les avantages sont :

- Pas de changement interne à Squash
- Options de création pour le fichier
- IHM
- Pas d'erreur si les consignes sont suivis

Néanmoins, cela reste un algorithme avec une récupération statique du texte dans le sens où je dois dire à l'algorithme quels mots garder.

3. Futures améliorations

Les améliorations à prévoir pour l'application serait de récupérer le texte selon les *run* tout en récupérant le style de celle-ci pour les garder dans le document final, j'ai fournis une documentation sur comment faire ceci sur mon GitHub.

La récupération dynamique du texte est aussi une amélioration prévu par Lumiplan, car si le les développeurs de Squash décide de changer la façon dont sont générés les rapports, l'application deviendra inutile. Cependant, aucune modification de ce Template n'est prévu dans la roadmap de Squash qui va jusqu'à fin 2017.

Une demande qui avait également été faite mais dont mon maitre de stage ne savait pas réalisable en dix semaines était de récupérer le texte de n'importe quelle source (docx, txt, XML) pour l'inclure dans le document final.

4. Mise à jour de Squash™

Lors de mon stage, j'ai fournis une documentation pour mettre à jour Squash à cette adresse.

Il faut donc procéder à une réinstallation complète de Squash excepté pour sa base de données MySQL si l'on veut garder les données.

Cependant il faut aussi mettre à jour la base de données car le modèle de données de la base change avec les versions de Squash. Pour cela, Squash met à disposition des scripts pour réaliser cette mise à jour.

Lors de l'avant dernier jour de mon stage, j'ai commis une erreur en supprimant un dossier de cache de Squash qui est censé se créer tout seul lors du lancement du serveur. Sauf qu'il m'était impossible relancer le dit serveur. J'ai donc procéder à la mise à jour avec l'accord de mon maitre de stage.

J'ai procéder comme dit dans ma documentation en réinstallant Squash et en exécutant la commande suivante :

```
# mysql -h HOST -u USER -p squashtm < /path/to/script.sql
```

Pour mettre à jour d'une version ma base de données. Tout s'est passé sans encombre.

Conclusion.

Mon stage dans l'entreprise c'est très bien passé, je suis content d'avoir trouvé un stage orienté dans le développement de logiciel, de plus c'était avec un langage de programmation qui me plaisait beaucoup. Toutes les personnes de l'entreprise ont été très gentilles avec moi, je me sentais comme un vrai salarié de Lumiplan. De plus les connaissances apprissent lors de mon DUT mon permissent d'avancer convenablement dans le sujet de mon stage.

Je pense que le fait que je sois autonome a été un vrai plus, j'ai pu avancer sur les parties qui étaient flous pour moi en faisant des recherches dessus, tout en demandant aux membres du BEL en cas de besoins sur leurs envies vis-à-vis de l'application. J'ai rencontré quelques moments bloquant pour la rédaction des scripts ou du Template de ma première application. Cependant, la diversité des choses à faire m'ont permis de ne pas horripilé d'être bloqué sur une seule chose. Je pense que le travail que j'ai réalisé au cours de ces dix semaines a été enrichissant. De plus les membres du BEL avaient l'air content du travail que je leur ai fournis même s'il reste des améliorations à faire. Cependant, Lumiplan compte bien continuer d'entretenir cette application en l'améliorent eux même ou par un stagiaire.

Glossaire

BEL : Bureau d'Études Logiciel

Exigences : les exigences sont l'expression d'un besoin documenté sur ce qu'un produit ou un service particuliers devraient être ou faire.

Cas de test : Chemin fonctionnel à mettre en œuvre pour atteindre un objectif de test. Un cas de test se définit par le jeu de test à mettre en œuvre, le scénario de test à exécuter et les résultats attendus. Il correspond à la mise en œuvre des exigences.

Template : Un Template est l'architecture d'une page, il représente la mise en forme que va avoir mon document final.

IHM : une Interface Homme Machine est l'ensemble des moyens utilisés par l'homme pour communiquer avec une machine.

Jetty : Jetty est un serveur HTTP et un moteur de servlet entièrement fondé sur la technologie Java. Squash étant un servlet, il est comme une application pour Jetty, grâce à cela, Squash peut être déployé sur un site web.

TomCat est également serveur HTTP et un moteur de servlet qui est cependant plus connu que Jetty.

Debian est un système d'exploitation basé sur des logiciels libre. Il est par conséquent gratuit et est comparable à Linux

SVN : ou subversion est un logiciel de gestion de versions, il permet de déposer ces fichiers sur un serveur, cela permet de garder une trace des différentes versions d'un logiciel. GitHub utilise également cette technologie.