UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CENTRALE D'ALERTE WEB / CENTRAL WEB ALERT (CWA)

RAPPORT DE PROJET

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN GÉNIE LOGICIEL

PAR

GILBERT YOUSSEF

AVRIL 2017

REMERCIEMENTS

En premier lieu, je veux remercier mes parents pour leurs grands encouragements tout le long de la maîtrise. À plusieurs reprises, j'ai voulu arrêter la poursuite de ma maîtrise à cause de l'effort et du temps que je devais y consacrer en plus de mon travail à temps plein. C'est grâce au soutien qu'ils m'ont offert que j'ai pu me rendre à destination.

Pareillement, je remercie mon directeur de projet, M. Guy Tremblay, pour son rôle de guide et ses conseils tout au long du projet qui m'ont aidé à trouver plusieurs solutions simples et efficaces.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Ce projet repose sur plusieurs aspects de développements de logiciel. Évidemment, l'objectif principal est de répondre à des besoins d'affaire de sorte qu'ils pourront être satisfaits de façon plus efficace, rapide, automatisée, efficiente et sûre. Le logiciel développé est un système de surveillance des tâches cédulées et exécutées par des processus ou des logiciels. Il génère des notifications lorsque des erreurs sont signalées et joue un rôle d'aide à la décision aux utilisateurs pour résoudre les problèmes survenus.

Un autre aspect traité dans ce projet est l'exploration de méthodes de développement qui reposent sur les principes agiles, d'une méthode d'estimation basée sur la complexité d'une tâche par phase de développement et l'utilisation de méthodes de suivis basés sur le graphique *sunset* et les indices de cohérences de leviers.

D'autres aspects, importants aussi, comme les outils de développements, d'intégration continue, d'automatisation et d'assurance qualité seront traités.

Les prochaines sections détaillent ces divers aspects et seront suivies d'une évaluation afin de tirer les leçons de cette expérience.

MOTS-CLÉS : Système de surveillance et de notification, méthodes de développement, méthodes d'estimation, intégration continue, assurance-qualité, outils de développement et automatisation.

INTRODUCTION

L'idée du projet provient d'un besoin que j'avais constaté durant mon travail dans une entreprise de courtier en services bancaires et d'investissement. Une des tâches principales et prioritaires dans le modèle d'affaire de l'entreprise est d'assurer le bon déroulement des processus hebdomadaires. On entend par processus un ensemble de tâches qui sont planifiées par une cédule dans "Windows Task Scheduler" sur plusieurs serveurs. Ces tâches déclenchent des Macros dans MS Access ou des scripts, où ces derniers téléchargent des données à partir des serveurs externes, transfèrent des données à l'interne et à des parties tierces, lancent des calculs, concilient des transactions, traitent des fichiers et génèrent de multiples rapports.

Le temps est très important dans cet environnement, des retards dans le transfert des données ou dans la génération des rapports n'affecte pas seulement, l'image, la productivité et les gains de l'entreprise, mais il peut emmener à des pénalités sévères et même à la fermeture de l'entreprise à cause de réglementations organisationnelles, gouvernementales et parfois même internationales.

Le problème avec le modèle actuel est que la surveillance des processus est manuelle et basée sur des fichiers sauvegardés, des documents imprimés en papier et des connaissances personnelles non partagées.

Dans ce qui suit, je présente sept chapitres pour expliquer en détail la problématique et la solution voulue, les fonctionnalités du logiciel, les méthodes de développement, les outils et l'environnement de développement, l'architecture du système, les étapes de développement et terminant par une évaluation du processus utilisé et du produit conçu.

CHAPITRE I

PROBLÉMATIQUES ET SOLUTIONS

Le but de ce chapitre est de présenter les problématiques rencontrées avec le système actuel ainsi que les solutions proposées.

1.1 Situation courante

La vérification des processus du système actuel se fait en se référant à un fichier Excel et un document écrit à la main. Des inspections sont faites de temps en temps pour comparer les courriels reçus avec la feuille d'Excel et s'assurer que les processus ont été exécutés avec succès.

De plus, parfois les processus s'exécutent avec succès, mais le statut réel est erroné parce que le courriel de confirmation n'a pas été reçu pour une raison X. En outre, de temps en temps un courriel saute de la vue ou un nouveau processus est mis en place mais le fichier Excel n'est pas partagé ou mis à jour.

D'ailleurs, chaque membre de l'équipe compte sur ses propres connaissances pour résoudre les problèmes au lieu de se référer à des documents d'appui. Il y a souvent des difficultés à résoudre un même problème qui était résolu auparavant, à cause des oublis.

En outre, souvent il y aura de temps perdu à chercher l'emplacement de l'application, des fichiers entrants ou extraits et des rapports générés dans le code source parce que les informations sont manquantes ou incorrectes dans les fichiers de référence.

Finalement, aucun partage des problèmes et solutions ne se fait avec les autres membres de l'équipe.

1.2 Solutions proposées

Étant donné que le système doit être surveillé en dehors des heures de travail normales, la meilleure façon est d'établir un site web sécurisé accessible à distance en tout temps à travers l'Internet. Le site web doit comporter la liste des tâches cédulées avec les détails intéressants (ex : statut, temps, interdépendances, fichiers entrants et sortants, erreurs connues et leurs solutions, références aux documents d'appui, personnes et départements impliqués, ...) qui permet de faire un suivi efficace et de résoudre les problèmes rapidement. De plus, il doit permettre de conserver les incidents et les erreurs dans une base de données afin qu'ils soient rapportés et d'établir des corrections en conséquence.

Une autre fonctionnalité importante est d'ajouter un outil de notifications qui peut générer des courriels et des alertes aux téléphones mobiles.

CHAPITRE II

FONCTIONNALITÉS DU LOGICIEL

Le but de ce chapitre est de donner une vue d'ensemble des fonctionnalités du logiciel ainsi que les principales interactions avec le logiciel.

2.1 Vue d'ensemble des fonctions du logiciel

Le coeur du logiciel est de *surveiller* des *processus (ou tâches)* qui s'exécutent selon une *cédule* spécifique et sur un *serveur dédié*. Ces processus appartiennent à des *modules* (un module comporte un ensemble des processus).

Les processus peuvent être *dépendants* d'autres processus avant de se déclencher. Pour gérer les dépendances entre les processus, chaque processus génère un fichier *flag* qui confirme qu'il s'est terminé avec succès. Le fichier est ensuite vérifié par les processus qui dépendent de lui.

Un *tableau de bord* est nécessaire pour afficher l'ensemble des processus avec de détails qui permettent de faire un suivi sur chacun d'eux (temps de démarrage et fin, statut et autres informations, etc.).

Il y a des *jours fériés* où les processus ne doivent pas s'exécuter.

En cas d'erreur avec un processus, il faut transmettre une notification à une personne *contact*, qui devra régler le problème.

De plus, il y a un intérêt à conserver les *historiques* des problèmes survenus, à avoir un *wiki* d'informations et à avoir un *graphe* qui montre les dépendances entre l'ensemble des processus.

Voici donc une liste qui résume les fonctionnalités du logiciel :

- Surveillance des processus.
- Gestion des processus.
- Gestion des serveurs.
- Gestions des cédules.
- Gestions des dépendances.
- Gestion des modules.
- Gestion des *flags*.
- Tableau de bord des processus.
- Gestion des contacts.
- Assignation d'un contact à un processus.
- Gestion des notifications.
- Gestion des jours fériés.
- Gestion d'historique des problèmes.
- Génération d'un graphe des dépendances.
- Gestion de *wiki*.

En général, le terme « gestion » concerne les opérations d'ajout, modification, suppression et de listage d'items.

D'autres fonctionnalités étaient nécessaires comme la gestion des usagers et de sécurité du site web, mais elles ont été omises parce que Liferay offre ces fonctionnalités ainsi que plusieurs autres.

2.2 Cas d'utilisation global

Pour l'entreprise pour laquelle le logiciel est conçu, l'équipe de développement et de support informatique (qui comporte trois analystes-programmeurs) est le seul *acteur* utilisateur du logiciel. C'est cet acteur qui établit, initialise et utilise le logiciel.

Cet acteur est indiqué par « administrateur » dans le diagramme de cas d'utilisation ci-bas.

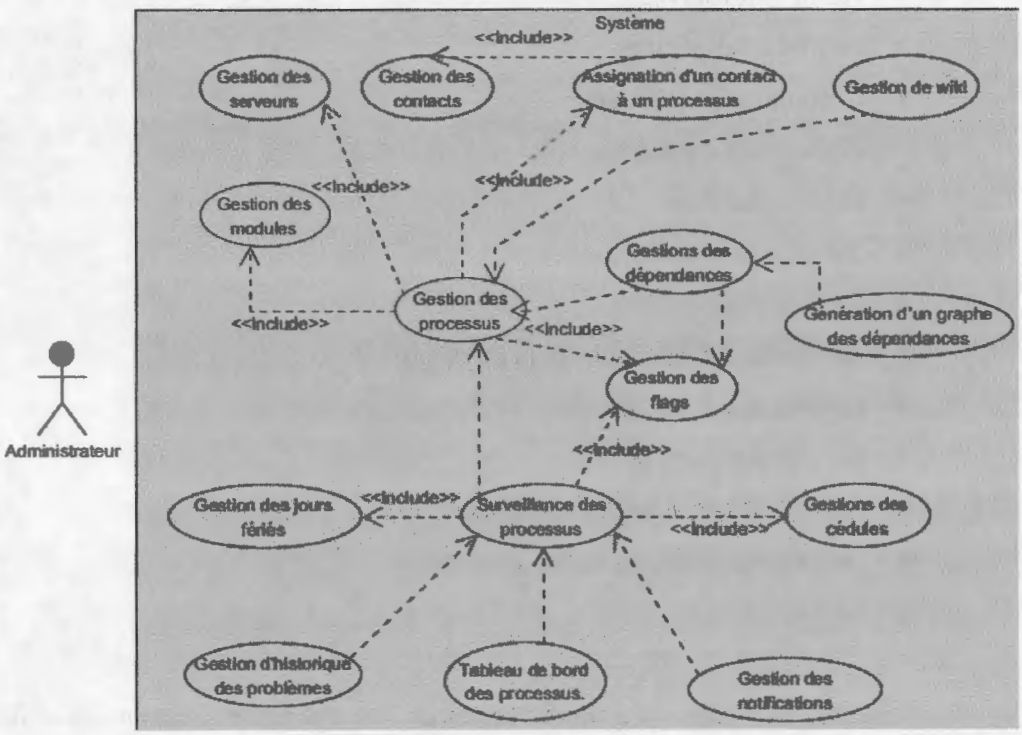


Figure 2.1 Diagramme de contexte

Les relations entre les cas d'utilisation sont comme suit :

- Une flèche en pointillé indique une relation de dépendance dans le sens de la flèche (gestion d'historique des problèmes, tableau de bord des processus et gestion des notifications dépendent de la surveillance des processus).

- Une flèche en pointillé avec la mention « include » indique une relation d'obligation (la gestion des processus possède une relation d'obligation avec la gestion des serveurs et la gestion des modules, parce que l'ajout d'un processus ne peut pas être fait sans l'existence d'un module parent et d'un serveur qui joue le rôle d'hôte). Après l'ajout d'un nouveau processus, il y a une obligation à créer un fichier *flag* pour ce processus.

2.3 Diagramme de classes

La Figure 2.2 présente le diagramme de classes de haut niveau qui montre l'ensemble des classes et les différentes relations entre celles-ci.

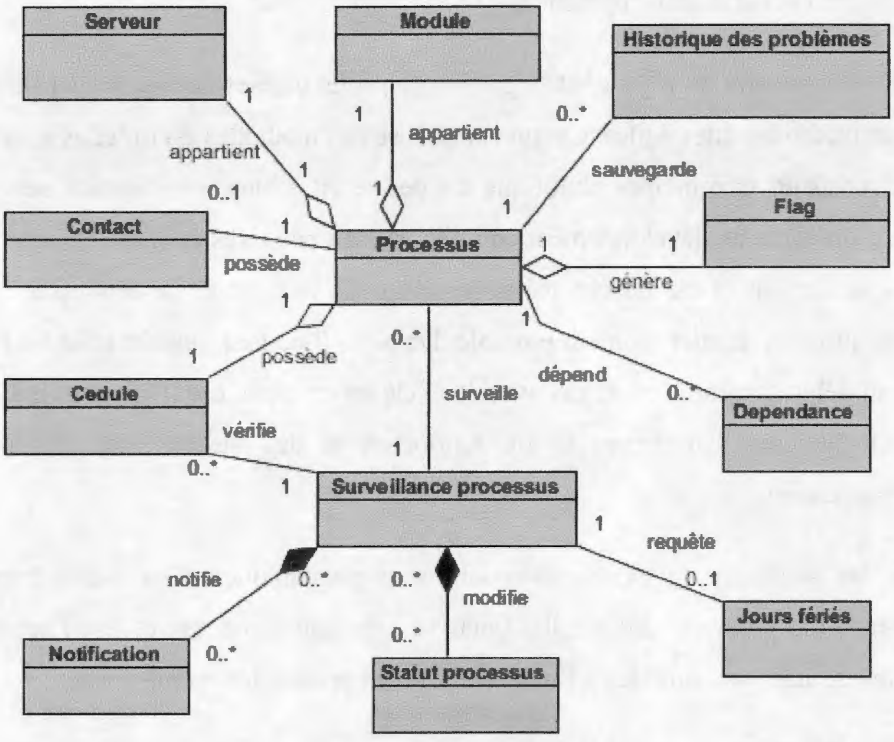


Figure 2.2 Diagramme de classes

CHAPITRE III

MÉTHODES DE DÉVELOPPEMENT

Le but de ce chapitre est de présenter les méthodes de développement utilisées et les divers processus de contrôle et du suivi.

3.1 Processus de développement

Le développement du logiciel et la gestion de projet utilisés dans ce projet sont basés sur les méthodes dites agiles. Ce qui caractérise ces méthodes est qu'elles sont basées sur des valeurs et principes plutôt que sur des recettes bien spécifiques à suivre. Les phases du cycle de développement en agile ne sont pas fixes et elles sont ajustées en continu. Un aspect clé de ces méthodes est qu'il faut tenter de repousser certains détails jusqu'au dernier moment possible. De plus, il ne faut plus détailler les besoins tout au début comme c'est le cas avec le cycle en cascade, car il est impossible qu'il n'y ait pas de changement et de repriorisation des besoins tout au long du développement.

Avec les méthodes agiles, les livraisons sont plus fréquentes et à des intervalles réguliers. Les phases traditionnelles (analyse, conception, codage et tests) ne sont pas éliminées, mais sont unifiées à l'intérieur d'un « sprint ». Un sprint permet :

- De détecter les anomalies et les problèmes le plus tôt possible afin de trouver des solutions convenables et d'éliminer l'accumulation des dettes techniques.
- D'avoir un retour rapide et fréquent du client.

- De connaître les coûts et le temps, parce que les sprints et l'équipe sont déterminés en avance. Ce sont les fonctionnalités à réaliser qui sont variables et adaptables en fonction des autres contraintes.
- De mesurer clairement et d'avoir un suivi de l'avancement.

3.1.1 Analyse des besoins et production du carnet de produit (*backlog*)

Les besoins sont exprimés sous forme de récits d'utilisateurs (*user stories*). La formalisation des récits doit répondre à trois questions importantes – qui? quoi? pourquoi? – afin d'identifier le plus clairement possible les fonctionnalités à réaliser.

Un exemple d'un récit d'utilisateur en lien avec notre projet : comme administrateur du système (spécifie le qui, la personne qui demande), je veux pouvoir ajouter une nouvelle tâche (spécifie le quoi, le besoin à développer ou la tâche à accomplir) à la liste des tâches afin de le surveiller (spécifie le pourquoi, le but du besoin).

Par la suite, ces récits sont priorisés et estimés comme nous le décrivons dans les sections qui suivent.

3.1.1.1 Priorisation des récits

Les récits sont priorisés selon trois niveaux :

- Critique : ces récits concernent les fonctionnalités à grande valeur pour l'utilisateur où le système ne peut pas être opérationnel sans qu'ils soient accomplis.
- Important : ces récits viennent en deuxième priorité et donnent une valeur ajoutée à l'expérience utilisateur, mais ne bloquent pas la mise en production du logiciel.

- Optionnel (*Nice to have*) : ces récits concernent des fonctionnalités qui sont bonnes à avoir si le temps le permet, selon la durée spécifiée pour la livraison en cours (*release*), mais qui pourraient être omises.

Par exemple, dans notre cas, le logiciel doit afficher en premier temps les statuts des processus sur une page web dédiée et envoyer des notifications par courriel lors des erreurs (récit de niveau critique). Par la suite, une notification par des messages téléphoniques courts (*SMS*) sera ajoutée (récit de niveau important).

3.1.1.2 Estimation des récits

L'estimation du temps et du coût d'un projet ou d'une tâche dans le domaine du développement logiciel reste un grand défi. C'est une des raisons principales de l'échec de plusieurs projets.

Plusieurs méthodes d'estimation ont été expérimentées dans mes projets précédents, mais aucune d'entre elles ne permet d'avoir une estimation exacte et fiable. L'estimation est toujours approximative et fournit un intervalle entre durée minimale et maximale.

Dans ce projet, j'ai pris la décision de m'éloigner des méthodes traditionnelles que j'avais utilisées dans divers projets passés. Après plusieurs recherches, j'ai trouvé une autre méthode d'estimation que j'ai voulu expérimenter.

Cette méthode d'estimation se base sur un ensemble d'indicateurs de complexité (Sanders, 2014). Les indicateurs sont répartis selon quatre niveaux de complexité : *Easy, Medium, Hard et Complex*. Ainsi, les pourcentages de consommation de temps sur les différentes phases de développement (*Design, Build, Test et Document*) ont été estimés. Ces estimations sont basées sur l'expérience et l'expertise vécues dans des projets précédents que j'avais accomplis et sont ajustables (voir Figure 3.1).

Complexity	Design (%)	Build (%)	Test (%)	Document (%)	Total Effort (%)	Total (Hours)
Easy	5%	83%	10%	2%	100%	4
Medium	10%	73%	15%	2%	100%	15
Hard	15%	59%	20%	6%	100%	25
Complex	20%	45%	25%	10%	100%	35
Special Case						

Design (Hours)	Build (Hours)	Test (Hours)	Document (Hours)	Total (Hours)
0.2	3.32	0.4	0.08	4
1.5	10.95	2.25	0.3	15
3.75	14.75	5	1.5	25
7	15.75	8.75	3.5	35

Figure 3.1 Indicateurs de complexité de l'estimation

Une fois que le tableau de complexité est prêt, il faut procéder à estimer les récits de la manière suivante. Pour chaque récit, il faut préciser le niveau de complexité pour trois catégories différentes. Encore une fois, les catégories, les phases et les indicateurs peuvent être modifiés selon le type et la complexité du projet à développer.

Dans mon projet, les catégories qui ont été choisies sont respectivement :

- IU : Interface utilisateur.
- Schéma/BD : Base de données.
- Intégration : Logique et intégration des tâches pour accomplir un besoin.

Après la saisie des niveaux de complexité pour chaque catégorie, l'estimation de temps est générée automatiquement (voir Figure 3.2).

A	B	C	D	E
Description	**UI**	**Integration**	**Schema/DB**	**Hours**
Comme utilisateur, je veux pouvoir ajouter, modifier ou détruire une tâche.	Hard	Hard	Complex	85

Figure 3.2 Estimation en heures d'un récit

Une fois que l'estimation pour l'ensemble des récits est terminée, l'effort requis pour accomplir le projet et le temps à consacrer aux différentes phases de développement peuvent être calculés (voir Figure 3.3).

Units of Work By Estimated Complexity	UI	Integration	Data Model	Total	Hours Modifier	Total Effort	
Easy	11	4	12	27	4	108	
Medium	1	9	4	14	15	210	
Hard	3	2	0	5	25	125	
Complex	2	2	1	5	35	175	
Special Case							
						618	Hours

Breakdown by Task	Easy	Medium	Hard	Complex	Totals		
Design	5.4	21	18.75	35	80.15		
Build	89.64	153.3	73.75	78.75	395.44		
Test	10.8	31.5	25	43.75	111.05		
Document	2.16	4.2	7.5	17.5	31.36		
Totals	108	210	125	175			
					618	Hours	

Figure 3.3 Estimation de l'effort pour accomplir le projet

De plus, le nombre de semaines ou jours requis selon différents horaires de travail peut être calculé (voir Figure 3.4).

Based on..		3	Hours per day
	Design	26.71666667	Days
	Build	131.8133333	Days
	Test	37.01666667	Days
	Documen	10.45333333	Days
Total		618	Hours
Total		206	Days

Based on...		15	Hours per week
	Design	5.343333333	Weeks
	Build	26.36266667	Weeks
	Test	7.403333333	Weeks
	Documen	2.090666667	Weeks
Total		618	Hours
Total		41.2	*Weeks*

Figure 3.4 Estimation adaptée aux horaires de travail

En outre, avant de commencer le travail sur un récit, il faut s'assurer qu'il répond à la règle INVEST. L'acronyme INVEST vient des termes suivants (Wake, 2003):

I : Independent, N : Negotiable, V: valuable, E: Estimable, S: Small, T : Testable

Cela veut donc dire de détailler le récit suffisamment pour qu'il soit testable, estimable, indépendant, clair et assez petit pour pouvoir être réalisé dans un sprint.

Id	Package	Parent	Type	Description	Priorité
14.2	Gestion des tâches	14	01-User Story	Comme utilisateur, je veux saisir des données dans une page pour pouvoir ajouter une nouvelle tâche	Critical
14.2.1	Gestion des tâches	14.2	02-Condition of satisfaction	Comme utilisateur, je dois recevoir un message de retour comme confirmation.	Critical
14.2.2	Gestion des tâches	14.2	03-Task	Valider les données saisies avant de les envoyer a la BD.	Critical
14.2.3	Gestion des tâches	14.2	03-Task	S'assurer que la tâche ajoutée n'existe pas (Nom tâche court unique).	Critical
14.2.4	Gestion des tâches	14.2	03-Task	gestion des path pour les applications avec liste déroulante	Critical
14.2.5	Gestion des tâches	14.2	03-Task	remplacer le champ d'entrée du nom du serveur par une liste déroulante	Critical
14.2.6	Gestion des tâches	14.2	03-Task	Documentation sur la tâche d'ajout tâches	Important
14.2.7	Gestion des tâches	14.2	03-Task	mise en production	Critical
14.2.8	Gestion des tâches	14.2	03-Task	Tests pour la tâche d'ajout tâches	Critical

Figure 3.5 Détailler un récit selon la règle INVEST

3.1.2 Développement en itération (sprint)

Le développement de projet est fait en phases itératives nommées sprint. La durée d'un sprint varie entre deux et cinq semaines. Dans le cas de notre projet, le choix est tombé sur des sprints de deux semaines pour une durée de 35 heures par sprint. Plusieurs raisons ont influé cette décision.

La première est que le tableau d'indices de complexité de l'estimation mis en place comporte une estimation de 35 heures pour une tâche complexe (voir Figure 3.3). La plus grande tâche doit donc être accommodée dans un sprint.

La deuxième raison est d'avoir plusieurs sprints courts, afin de détecter les problèmes le plus tôt possible et faire les ajustements nécessaires.

De plus, le but est d'avoir plus de données de comparaisons entre les différents sprints. Différentes données sont générées à chaque fin de sprint, comme la vélocité de l'équipe, les pourcentages d'avancement et de consommation du budget.

Un but à accomplir est spécifié à l'avance pour chaque sprint. Il comporte un nombre de récits qui sont extraits du carnet de produit et ajoutés au sprint courant, selon le niveau de priorité des récits. Les récits de haute priorité sont traités en premier.

Finalement, le projet est décomposé sur plusieurs livraisons nommées *release*. Chaque livraison comporte un nombre prédéfini de sprints. La première livraison comporte les fonctionnalités requises et critiques au besoin de travail. Les fonctionnalités moins importantes et les améliorations suivent dans d'autres livraisons.

3.2 Contrôle et suivi d'avancement

Pour s'assurer que le projet suivre le trajet voulu, il faut identifier les problèmes le plus tôt possible. Pour cela, il faut mesurer de temps en temps, comparer les indices de coût, d'effort et les fonctionnalités accomplies et apporter les ajustements nécessaires au besoin.

Plusieurs procédures, mesures et indicateurs ont été mis en place pour faciliter le contrôle et le suivi d'avancement du projet, notamment une rétrospective et une revue faites à chaque fin de sprint. Plusieurs indicateurs d'avancement sur le coût, le temps et l'effort ont été capturés. Finalement, un graphique d'avancement global nommé *Sunset Graph* a été mis à jour régulièrement. Ce graphique est riche, fiable et donne

une vue sur l'avancement du projet. Ces derniers items seront détaillés dans les sous-sections qui suivent.

3.2.1 Indicateur d'avancement d'un Sprint (*Sprint Burndown*)

Avant de commencer un sprint, une estimation additionnelle et raffinée est faite sur les tâches et sous-tâches d'un récit.

Id	Package	Parent	Type	Description	Status	Original Estimate	Day 1	Day 2	Day 3	Day 4	Day5	Day6	Sprint Review
23.2	Gestion des incidents	23	03-Task			15	15	15	0	0	0	0	0
..	...	...	...	...		...	...	...	...	...	...	...	...
			Total			35							

Figure 3.6 Estimation raffinée d'un récit

Cette nouvelle estimation permet d'avoir une vue plus claire et transparente de l'avancement tout le long d'un sprint. L'avancement est exposé par un graphique nommé *Sprint Burndown*.

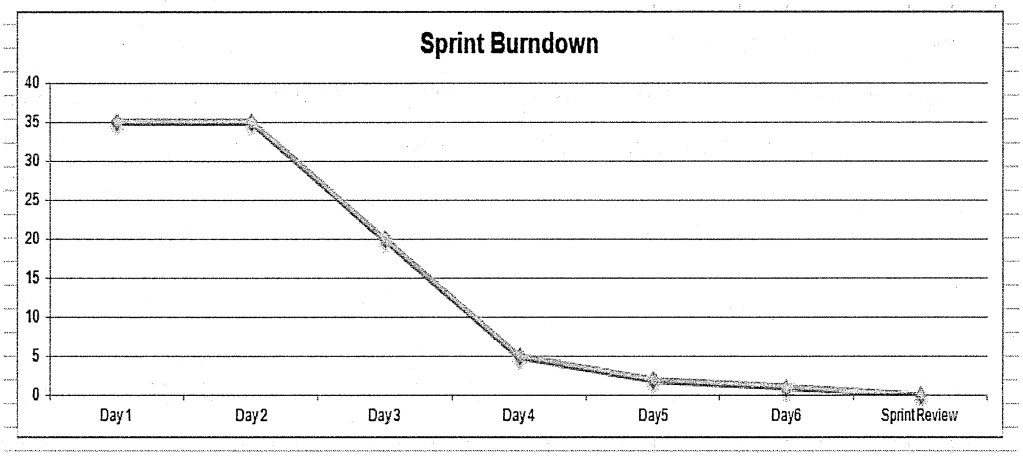


Figure 3.7 Graphique d'avancement d'un sprint (*Sprint Burndown*)

3.2.2 Revue de sprint

Une révision a été faite à la fin de chaque sprint. Son but était de comparer ce qui a été accompli par rapport à ce qui avait été planifié et d'apporter les ajustements nécessaires. C'est le produit final qui est mis en question, le résultat, pas « le comment ». C'est donc une vérification des fonctionnalités du logiciel accomplies qui est faite en comparant les résultats avec les récits et leurs critères de succès.

3.2.3 Rétrospective

Une rétrospective a été faite à chaque fin de sprint. Son but est spécifiquement d'analyser le « comment ». Il faut regarder quelles méthodes, techniques et pratiques ont été utilisées durant le sprint afin de les évaluer et d'apporter les corrections nécessaires.

3.2.4 Indicateurs d'avancement des sprints

À chaque fin de sprint, les nombres de points de fonction (la valeur de l'estimation assignée à un récit : dans ce projet c'est le nombre d'heures requis pour accomplir un récit) accomplis ont été enregistrés et organisés par niveau de priorité des récits. Par la suite, un calcul est généré automatiquement et présente plusieurs indicateurs (voir Figure 3.9).

Vélocité moyenne :

C'est la valeur moyenne des points de fonction que l'équipe de développement peut accomplir pendant un sprint. Cette valeur est utile pour connaître la performance de l'équipe et l'avancement du projet. Elle est utile pour les estimations de projets futurs.

Tendances minimales et maximales estimées :

Les tendances sont calculées en se basant sur les valeurs de la vélocité moyenne des trois sprints précédents. Elles donnent les tendances minimales (pessimistes) et maximales (optimistes) des fonctionnalités qui peuvent être réalisées pendant le temps alloué. C'est une prévision pour les sprints futurs.

Cohérence des leviers :

En basant sur les résultats des points de fonction accomplis à chaque fin de sprint, les indicateurs de coût, temps et fonctionnalités sont notés. Ces trois indicateurs (leviers) interreliés forment ensemble une synthèse de l'avancement et du statut du projet.

«*L'objectif est d'avoir un même niveau pour les trois à un moment donné (exemple : au sprint 5, si le budget est consommé à 50%, on s'attendrait à avoir un avancement à 50% du backlog, et que le projet soit en 10 sprints (i.e. sprint 5 = 50% du temps consommé)*» (Pyxis, 2015).

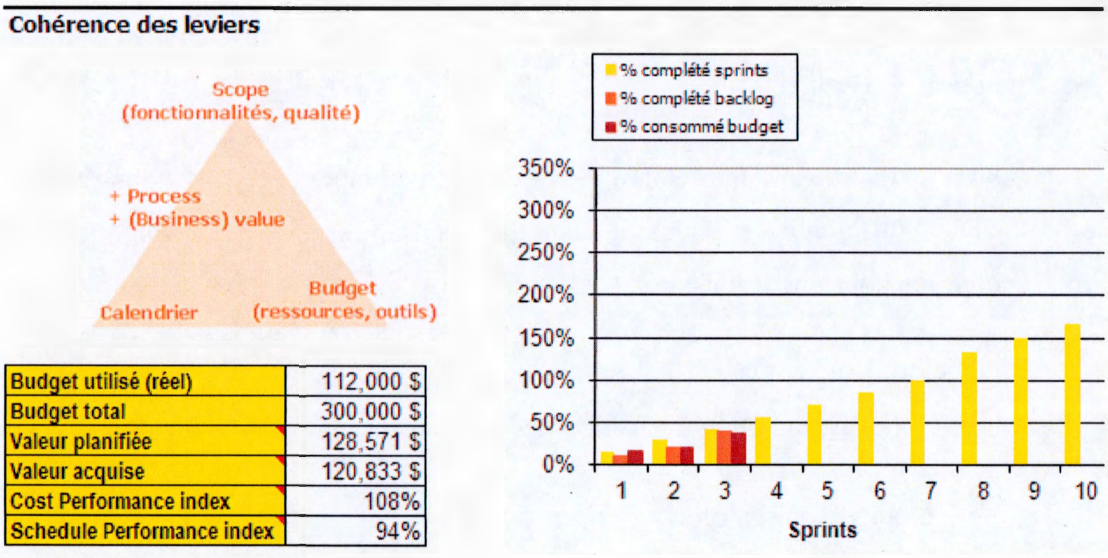


Figure 3.8 Cohérence des leviers

3.2.5 Sunset Graph

Ce graphe, conçu par la compagnie Pyxis (Pyxis, 2015), génère une vue globale et détaillée sur l'avancement du projet.

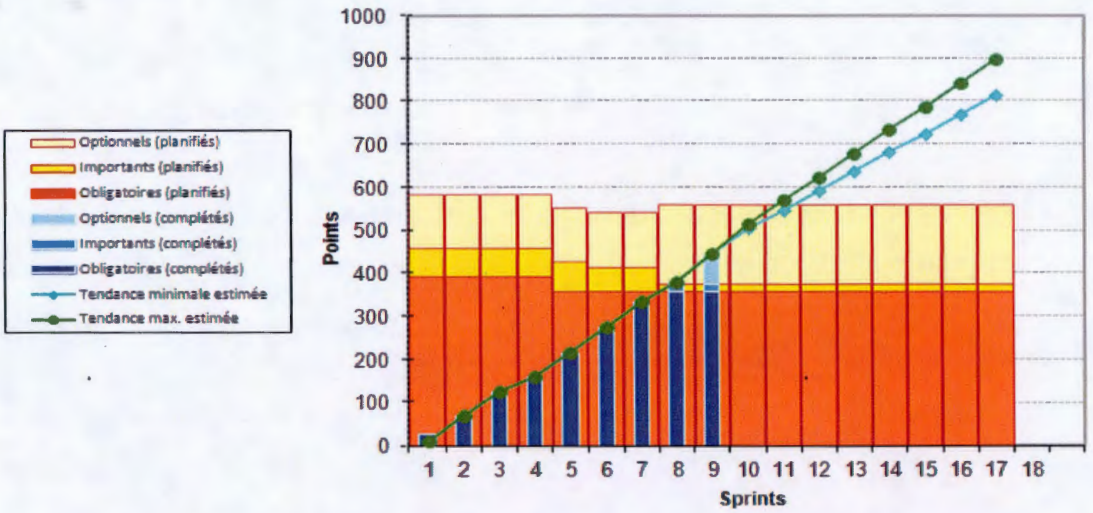


Figure 3.9 Tendances et avancement d'une livraison (Sunset Graph)

Le tableau est à deux dimensions, soit les sprints en horizontal et les points de fonction en vertical.

Pour chaque sprint, un rectangle est dessiné comportant trois sous-rectangles de couleurs différentes. Ce rectangle indique l'ensemble des récits planifiés pour être accomplis pour une livraison.

Chaque couleur représente le nombre de points de fonction des récits planifiés selon les différents niveaux de priorisation des récits utilisateurs comme suit :

- Orange : obligatoire.
- Ocre jaune : important.
- Jaune : optionnel.

Un autre rectangle est dessiné à la fin de chaque sprint comportant trois sous-rectangles de couleurs différentes. Ce rectangle représente le nombre de points de fonction des récits complétés.

Chaque couleur représente le nombre de points de fonction des récits complétés selon les différents niveaux de priorisation des récits utilisateurs comme suit :

- Bleu foncé : obligatoire.
- Bleu marine : important.
- Bleu ciel : optionnel.

Les deux diagonales ascendantes (bleu ciel et vert) forment les tendances minimales et maximales estimées.

Les deux rectangles et les deux diagonales sont générés automatiquement après chaque enregistrement ou modification aux données du *backlog (*voir Figure 3.9).

Consolidation des données du backlog et du budget

Release	Sprint	Obligatoires (complétés)	Importants (complétés)	Optionnels (complétés)	Obligatoires (planifiés)	Importants (planifiés)	Optionnels (planifiés)	Vélocité moyenne	Tendance minimale estimée	Tendance max estimée
Release 1	1	28	0	0	391	67	127	28	10	10
Release 1	2	70	0	0	391	67	127	42	70	70
Release 1	3	125	0	0	391	67	127	55	125	125
Release 1	4	160	0	0	391	67	127	35	160	160
Release 1	5	215	0	0	359	67	127	55	215	215
Release 1	6	275	0	0	359	55	127	60	275	275
Release 2	7	333	0	0	359	55	127	58	333	333
Release 2	8	359	17	4	359	17	184	47	380	380
Release 2	9	359	17	69	359	17	184	65	445	445
Release 2	10				359	17	184	0	504	517
Release 2	11				359	17	184	0	548	571
Release 2	12				359	17	184	0	593	625
Release 3	13				359	17	184	0	637	680
Release 3	14				359	17	184	0	682	734
Release 3	15				359	17	184	0	726	788
Release 3	16				359	17	184	0	771	843
Release 3	17				359	17	184	0	815	897
Release 3	18									

Figure 3.10 Données du backlog

CHAPITRE IV

OUTILS ET ENVIRONNEMENT DE DÉVELOPPEMENT ET DE DÉPLOIEMENT

Le but de ce chapitre est de présenter les choix technologiques adoptés pour le développement du logiciel, les divers outils utilisés ainsi que les méthodes d'automatisation et de déploiement.

4.1 Choix technologiques

Les technologies utilisées dans ce projet ont été choisies en prenant en considération plusieurs facteurs. Essentiellement, l'environnement technologique en place dans la compagnie pour laquelle le logiciel était développé pour être utilisé, et de façon à ce que le logiciel soit le plus possible générique (multiplateforme et simple à être modifié et maintenu).

Les choix principaux ont été les suivants :

- Java : langage de programmation du *backend.*
- Java Server Page, JQuery, JavaScript, DataTable : langages de programmation du *frontend* et de tout ce qui est présentation.
- SQL Server : serveur de base de données. À noter que le logiciel est conçu de façon qu'il soit fonctionnel avec différents types de base de données.
- Liferay : gestionnaire de contenu web. Il possède plusieurs librairies utiles qui facilitent et accélèrent le développement du logiciel (gestion des usagers et

sécurité, gestion de documents, *wiki,* automatisation des tâches, bootstrap pour avoir un meilleur style pour les pages web et plusieurs autres).

- Tomcat : serveur web utilisé en développement, mais n'importe quel serveur web pourrait être utilisé en production.

Après l'identification des principaux choix technologiques, la prochaine étape était de spécifier les outils de développement, de gestion des versions et d'automatisation. À noter que plusieurs outils n'ont pas été utilisés dès le départ, mais plutôt ont été intégrés après avoir détecté des problèmes lors de revues et rétrospectives faites à chaque fin de sprint.

4.2 Conservation du code et gestion de versions

Bitbucket a été choisi pour conserver le code source du logiciel et pour gérer les versions. Bitbucket offre un service sans frais et comporte un ensemble d'outils utiles et faciles à utiliser pour automatiser les processus d'intégration.

À chaque fois qu'un récit était complété et à chaque fin de sprint, un commit était fait pour enregistrer le code source dans une branche appelée développement dans Bitbucket.

Une caractéristique importante de Bitbucket est son intégration avec Jenkins (serveur d'intégration continue). À chaque fois qu'un commit est fait, Jenkins détecte ce changement, lance une régénération de code (*build*) et exécute les tests pour s'assurer que le logiciel reste fonctionnel.

4.3 Outils et automatisation du processus de développement

- Maven : outil facilement intégré avec Éclipse qui s'occupe de tous les processus de compilation et de déploiement, de la gestion des librairies et de l'exécution des tests unitaires.
- Jenkins : serveur d'intégration continue qui permet d'automatiser les processus de compilation, de déploiement et de tests.
- Selenium et TestNG : Selenium est un outil d'automatisation de tests pour les applications web. Il permet d'enregistrer les étapes et les actions effectuées et de les reproduire. C'est un outil crucial pour s'assurer de la stabilité et du bon fonctionnement du logiciel. TestNG est un outil de tests unitaires. Il est semblable à Junit, mais plus avancé pour la création des tests en groupe, interdépendants et exécutés selon un ordre spécifique.
- Jautodoc : outil qui génère automatiquement la documentation sur le code (*JavaDoc*). Beaucoup de temps peut être sauvé avec cet outil si les noms des méthodes et classes respectent certaines conventions.
- Quartz Scheduler : librairie intégrée dans Liferay qui permet d'exécuter des tâches à des intervalles spécifiques et d'une façon asynchrone.
- EhCache : *cache* robuste utilisée pour avoir une bonne performance du logiciel et réduire la communication avec la base de données.
- SonarQube : outil d'assurance qualité qui permet de détecter les anomalies au niveau du code (duplication, complexité, manque de documentation ou tests, style et règles de codage, etc.).
- Graphviz : outil qui permet de représenter une structure d'information sous forme d'un graphe. Dans ce projet, le besoin est de représenter les interdépendances entre les tâches.
- Log4J : librairie qui permet d'enregistrer les erreurs et les événements du logiciel.

- Datatable : librairie JQuery pour représenter les données sous forme d'une table et les gérer dynamiquement.
- Formateur de code : assure que le code est uniforme et évite les ajustements faits à la main.
- Plateforme Spring : utilisée pour ses fonctionnalités d'initialisation du contexte de l'application, *BeanFactory,* et pour l'injection de dépendances.

4.4 Déploiement du logiciel en production

Pour déployer une nouvelle version du logiciel sur un serveur, il suffit d'accomplir les tâches suivantes :

- Avoir une copie du jar généré par Jenkins.
- Avoir une copie de la structure de la base de données si celle-ci est changée.
- Avoir une copie du site web qui est généré par Liferay s'il y a eu des changements (ajout de page web ou composante visuelle).
- Restaurer la base de données et le site web sur le serveur en question au besoin.
- Déposer le jar dans le répertoire nommé *deploy* sur le serveur en question.

CHAPITRE V

ARCHITECTURE DU SYSTÈME

Le but de ce chapitre est de présenter les choix architecturaux adoptés, les divers patrons de conception utilisés ainsi que les principales interactions entre les composants du système.

5.1 Choix architecturaux

Essentiellement, l'architecture utilisée est l'architecture MVC (Modèle, Vue, Contrôleur). Cette architecture sépare un logiciel en trois composants comme suit :

- Modèle : traite la logique métier (*business logic*) et la communication avec la base de données.
- Vue : responsable de la couche interface-utilisateur et de l'affichage des données.
- Contrôleur : traite les requêtes et les réponses en utilisant les données appropriées.

La raison principale de ce choix est de faciliter la maintenance du logiciel. Cela permet de remplacer ou faire une mise à niveau d'un des composants sans affecter les deux autres. Par exemple, des changements au niveau de la présentation peuvent être faits sans affecter le reste du logiciel. De même, un changement de type de base de

données peut être simple à faire en peu de temps, sans interférer avec les autres aspects.

Il faut mentionner qu'un des avantages de l'utilisation de Liferay est qu'il vient avec une plateforme MVC prête à utiliser pour développer des éléments graphiques.

La figure ci-bas (Figure 5.1) montre comment les composants sont séparés :

- Un contrôleur est créé pour chaque fonctionnalité (rectangle sur l'image).
- Chaque fonctionnalité possède son composant de modèle (triangle sur l'image).
- Chaque fonctionnalité possède les ressources visuelles nécessaires dans un composant séparé (cercle sur l'image).
- Toutes les fonctionnalités utilitaires sont séparées des autres composants (*Gprahviz*, *Quartz*, *cache*, connexion avec la base de données, etc.).

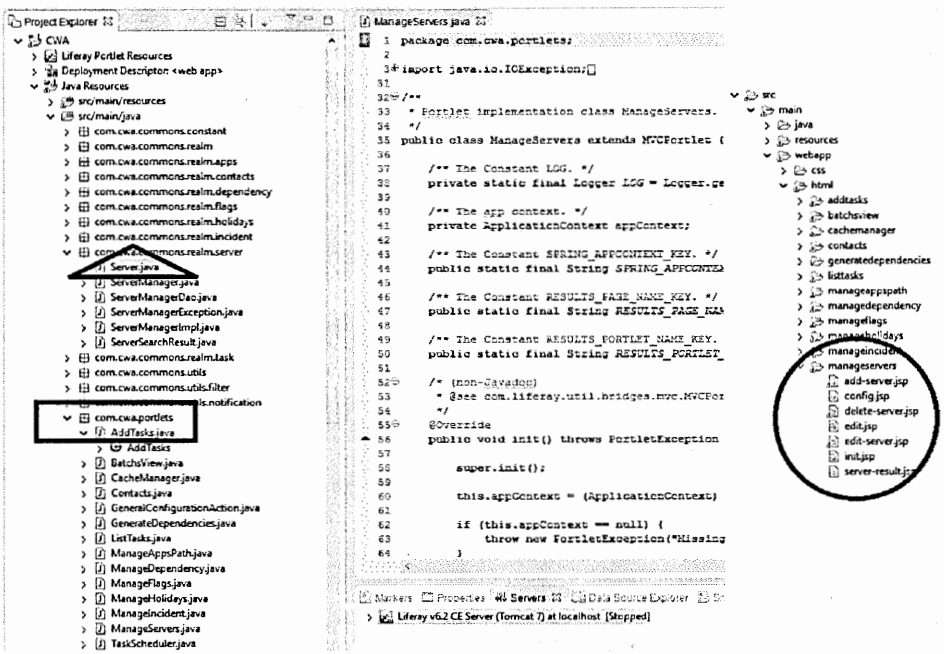


Figure 5.1 Architecture MVC

5.2 Diagramme de composants

La figure ci-bas (Figure 5.2) présente le diagramme de composants qui montre les associations et les dépendances entre les différents composants (paquetages, base de données, librairies, fichiers sources, configurations et activités).

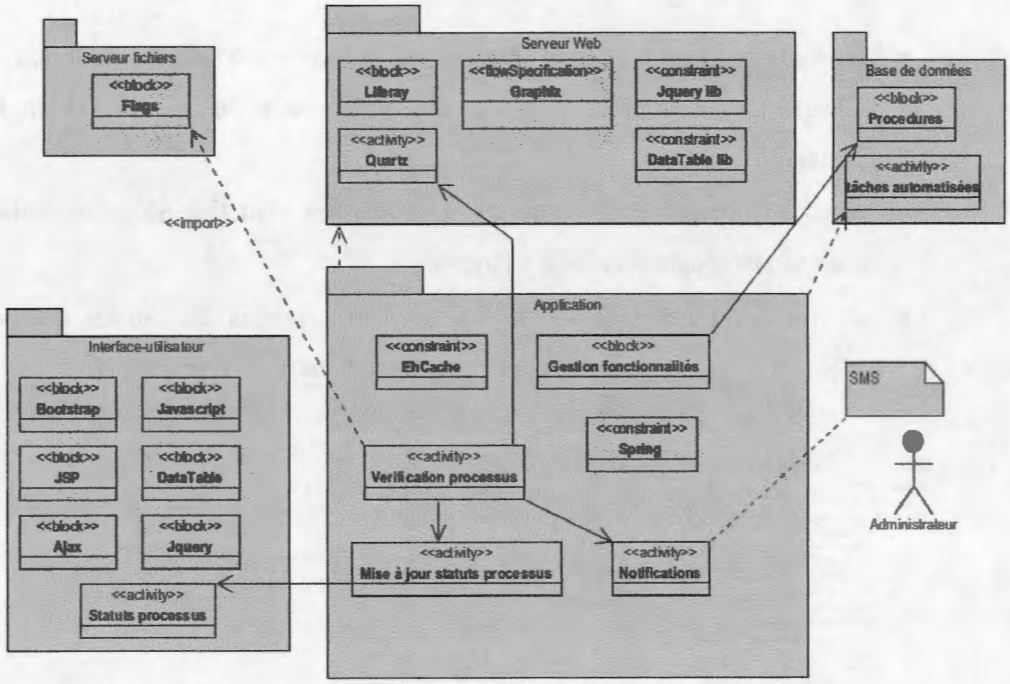


Figure 5.2 Diagramme de composants

- Une flèche noire indique qu'il y a une association de référence (une dépendance fonctionnelle sur le bloc référé) à un bloc ou une activité spécifique.
- Une flèche en pointillé indique une référence logique entre deux composants (une dépendance logique peut affecter les données et le comportement du système, mais non pas son fonctionnement).

- Le mot « *import* » sur une flèche indique qu'il existe une dépendance inclusive (le bloc référé fait partie du bloc original, mais séparé logiquement) sur un autre bloc.

5.3 Patrons de conception

D'autres patrons de conceptions que celui de MVC ont été utilisés dans ce projet :

- Injection de dépendances :

  Ce patron permet d'injecter des classes spécifiques lors de l'initialisation de l'application qui sont facilement modifiables au besoin. Cela donne un haut degré d'extensibilité et de maintenabilité à l'application.

```
<bean id="BdConnectorParamFormat" class="com.cva.services.dao.ParamFormatterImpl"
    scope="singleton" lazy-init="true">
</bean>

<bean id="com.cva.services.dao.BdConnector" class="com.cva.services.dao.GenericBDConnector
    scope="singleton" lazy-init="true">
    <constructor-arg>
        <value>java:comp/env/jdbc/CWADB</value>
    </constructor-arg>
    <constructor-arg>
        <ref bean="BdConnectorParamFormat"></ref>
    </constructor-arg>
</bean>
public GenericBDConnector(String jndiPath, ParamFormatter formatter) throws SQLException {

    if (jndiPath == null) {
        throw new IllegalArgumentException("Argument 'jndiPath' cannot be null !");
    }

    if (formatter == null) {
        throw new IllegalArgumentException("Argument 'formatter' cannot be null !");
    }

    this.formatter = formatter;

    this.initialize(jndiPath);

}

public class ServerManagerImpl implements ServerManager {
public ServerManagerImpl(ServerManagerDao dao) {
```

Figure 5.3 Injection de dépendances

L'exemple présenté dans la figure ci-haut montre comment on peut définir le lien vers la base de données à utiliser (rectangle sur l'image) et sous quel formatage (cercle sur l'image).

Ceci permet de modifier plusieurs paramètres qui sont spécifiques à l'utilisation du logiciel dans différents contextes (type de base de données, formatage de données selon le pays ou le langage, implémentation des méthodes de gestion des entités).

Les classes utilisées pour la gestion des entités sont toujours des classes qui implémentent des interfaces, afin de permettre de changer les implémentations au besoin, et ce simplement en changeant les paramètres dans le fichier de configuration (voir la flèche sur l'image de la Figure 5.3).

Les configurations des dépendances sont ajoutées à un fichier de type XML. L'application les initialise au déploiement de l'application sous forme d'un contexte d'application (voir le rectangle sur l'image ci-bas). Par la suite, chaque contrôleur les accède en se référant au contexte de l'application (voir le cercle sur l'image) (voir Figure 5.4).

```
public class MyPluginListener extends PluginContextListener {

    /** The Constant SPRING_APPCONTEXT_KEY. */
    public static final String SPRING_APPCONTEXT_KEY = "spring.app.context";

    /** The Constant SPRING_APPCONTEXT_XML_PATH. */
    public static final String SPRING_APPCONTEXT_XML_PATH = "ext-spring.xml";

    /*
     * (non-Javadoc)
     *
     * @see
     * com.liferay.portal.kernel.servlet.PluginContextListener#contextInitialized(javax.servlet.ServletContext
     */
    @Override
    public void contextInitialized(javax.servlet.ServletContextEvent servletContextEvent) {

        ApplicationContext context;

        context = new ClassPathXmlApplicationContext(new String[] { SPRING_APPCONTEXT_XML_PATH });

        servletContextEvent.getServletContext().setAttribute(SPRING_APPCONTEXT_KEY, context);

    }

}
```

ManageServers.java

```
35 public class ManageServers extends MVCPortlet {
36  @Override
37  public void init() throws PortletException {
38
39      super.init();
40
41      this.appContext = (ApplicationContext) this.getPortletContext().getAttribute(SPRING_APPCONTEXT_KEY);
42
```

Figure 5.4 Configuration du contexte de l'application

- Patron d'accès aux données des objets (*Data Access Object Pattern*) :
  Ce patron introduit un objet qui joue un rôle intermédiaire entre deux niveaux, le niveau qui demande d'accéder aux données et le niveau qui manipule les données.

```
public interface ServerManager {
    public void deleteServer(String serverId) throws ServerManagerException;
   ......
public class ServerManagerImpl implements ServerManager {

    /** The dao. */
    ServerManagerDao dao;
    public ServerManagerImpl(ServerManagerDao dao) {
        this.dao = dao;
    }
    @Override
    public void deleteServer(String serverId) throws ServerManagerException {
        try {
            dao.delete(serverId);
        } catch (DaoException e) {
            throw new ServerManagerException(ServerManagerException.SYSTEM_ERROR, "Error calling ServerManagerDao", e);
        }
    }
    .......
public class ServerManagerDao extends Dao<Server, RealmSearchParams, RealmSearchResult<String, Server>> {

    /** The Constant SERVER_ID. */
    private static final String SERVER_ID = "Server_Id";

    /**
     * Instantiates a new server manager dao.
     *
     * @param connector the connector
     */
    public ServerManagerDao(BdConnector connector) {
        super(connector);
    }
    public void delete(Server entity) throws DaoException {
     .......
public class Server implements Serializable {
    private String serverId;


    private String serverName;
    public String getServerName() {
        return serverName;
    }
    .......
```

Figure 5.5 Patron d'accès aux données des objets

L'interface *ServerManager* définit les opérations de façon abstraite et la classe *ServerManagerImpl* détient la référence à la base de données afin de demander l'exécution des opérations requises. L'exécution finale des opérations se fait dans la classe responsable de la communication avec la base de données (*ServerManagerDao*).

- Patron de transfert d'objets (*Transfer Object Pattern*) :

  Ce patron consiste à créer un objet de transfert (une classe avec des méthodes qui permettent d'accéder et de changer les données d'un objet, la classe nommée *Server* dans la figure ci-bas) dans le but de passer des données multiples en un seul appel (voir la méthode *parseAppFromRequest* qui transforme un ensemble d'attributs en un objet *Server*). Cet objet sera utilisé par les demandeurs de service afin d'exécuter des opérations sur l'objet. Ce même objet sera utilisé par le service désigné pour y répondre (voir les opérations de l'interface *ServerManager* et comment elles utilisent l'objet *Server*).

```
private Server parseAppFromRequest(Map<String, String[]> parameterMap) {
    Server formParams;

    formParams = new Server();
    formParams.setServerId(parameterMap.get(Server.PARAM_SERVERID_KEY) == null
                           ? null : parameterMap.get(Server.PARAM_SERVERID_KEY)[0]);

    formParams.setServerName(parameterMap.get(Server.PARAM_SERVERNAME_KEY) == null
                             ? null : parameterMap.get(Server.PARAM_SERVERNAME_KEY)[0]);
    return formParams;
}

public class Server implements Serializable {

    private String serverId;

    private String serverName;

    public String getServerName() {
        return serverName;
    }
    public void setServerName(String serverName) {
        this.serverName = serverName;
    }
    public String getServerId() {
        return serverId;
    }
    public void setServerId(String serverId) {
        this.serverId = serverId;
    }
}
public interface ServerManager {
 public Server addServer(Server serverParams) throws ServerManagerException;
 public RealmSearchResult<String, Server> getServers() throws ServerManagerException;
   ......

}
```

Figure 5.6 Patron de transfert d'objets

CHAPITRE VI

PROGRESSION DU PROJET ET PROBLÈMES RENCONTRÉS

Le but de ce chapitre est de présenter comment s'est fait l'avancement d'un sprint à un autre, et quelles ont été les problématiques rencontrées ainsi que les solutions adoptées.

6.1 Avancement d'un sprint à un autre

Les premiers trois sprints ont été plus exploratoires, où plusieurs changements étaient nécessaires afin de stabiliser l'architecture, les processus et les divers composants du logiciel. Plusieurs refactorisations ont été faites au deuxième sprint suite à la rétrospective et la revue faites à la fin du premier sprint. La décision sur le développement des tests et la rédaction de la documentation n'était pas trop claire à ce moment, s'il fallait les faire au fur et à mesure ou attendre que les fonctionnalités soient plus stables afin de sauver du temps.

Ensuite, pour les sprints suivants, l'accent était plus sur la qualité du logiciel. Des techniques avancées ont été utilisées pour avoir un haut niveau de performance du logiciel, comme l'utilisation du *cache* et les appels *Ajax* pour rafraîchir les statuts des processus sans rafraîchir toute la page HTML.

De plus, un formatage de code et l'utilisation de l'outil SonarQube ont été mis en place pour détecter les problèmes reliés à la qualité du code et du logiciel.

La figure ci-bas (Figure 6.1) montre les dettes techniques qui ont été découvertes lors de la première utilisation de l'outil SonarQube pendant le cinquième sprint.

Figure 6.1 Indices SonarQube avant les corrections

À partir du cinquième sprint, une partie du temps de chaque sprint a été consacrée à résoudre les problèmes de qualité. À la fin du neuvième sprint, l'ensemble des anomalies avaient été résolues (voir Figure 6.2).

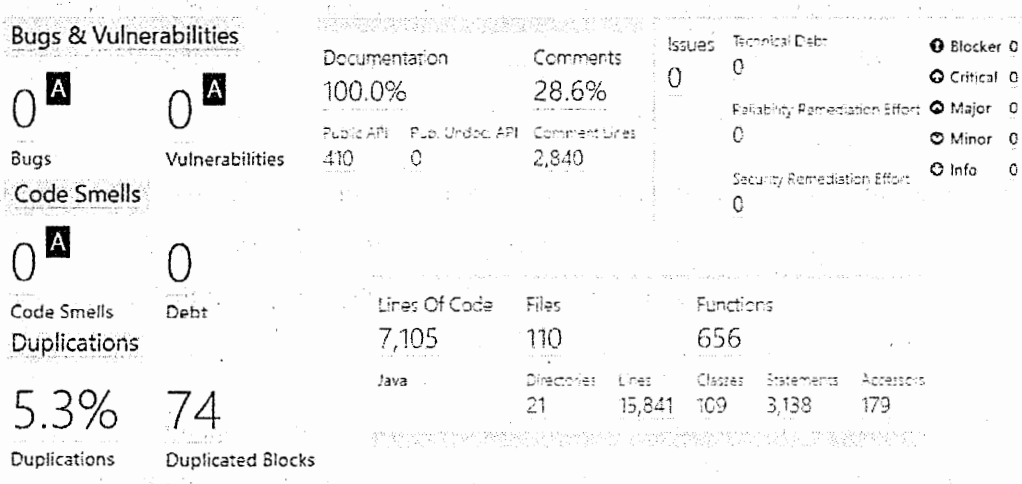


Figure 6.2 Indices SonarQube après les corrections

Finalement, il est important de mentionner que le développement était intuitif et simple à partir du cinquième sprint. Ceci est dû à la familiarité avec les diverses fonctionnalités du logiciel et l'architecture mise en place.

Voir l'appendice A pour plus de détails sur les problèmes identifiés durant chaque sprint.

6.2 Problèmes de dépendances internes et externes

Plusieurs problèmes ont été rencontrés durant le développement du logiciel, notamment certains problèmes « normaux », comme l'intégration des librairies ou des outils et leur utilisation.

Il y avait aussi d'autres problèmes importants qu'il est intéressant de souligner.

D'une part, il y avait des problèmes de dépendances internes. Par exemple, les tests développés par Selenium et TestNG utilisent les librairies du fureteur Web Firefox pour exécuter les tests automatisés. Comme le fureteur Web se met à jour automatiquement, il y a eu à un moment un problème de compatibilité entre les librairies. Plusieurs heures ont été consacrées afin de régler ce problème.

D'autre part, il y avait des problèmes de dépendances externes. Le serveur d'intégration Jenkins dépend du logiciel de gestion des versions Bitbucket. Ce dernier a été mis à jour à un moment et cela a causé un problème d'authentification pour Jenkins. Plusieurs heures ont été aussi nécessaires afin de régler ce problème.

Ceci emmène à l'importance de faire attention à l'utilisation des logiciels, leurs configurations et leurs interdépendances. Que ce soit des dépendances internes ou externes, il faut prendre en considération ce risque et le limiter à l'avance le plus que possible.

CHAPITRE VII

ÉVALUATION DU PROCESSUS ET DU PRODUIT

Le but de ce chapitre est d'évaluer le processus suivi et les méthodes utilisées dans le développement du projet ainsi qu'évaluer le produit final.

7.1 Évaluation du processus

Le projet avait été estimé pour être complété en 17 sprints (35 heures par sprint). Cette estimation avait été calculée en utilisant la méthode d'estimation basée sur des indices de complexité.

Durant le développement du projet, il y a eu nécessité de modifier, supprimer et ajouter de nouveaux récits utilisateurs. Par exemple, deux tâches ont été évaluées avec un indice de haute complexité lors de l'estimation, mais leur accomplissement était plus simple grâce à l'utilisation des outils ou des librairies choisis.

D'un autre côté, il y avait des tâches qui n'avaient pas été prises en considération lors de la planification, donc elles ont été ajoutées plus tard. Essentiellement, il s'agissait de tâches pour assurer la qualité du code et du logiciel.

Après neuf sprints de développement, le statut du projet est comme suit :

- Tous les récits obligatoires et importants ont été accomplis.
- Selon les estimations actuelles, les récits optionnels restants seront complétés au sprint 11 ou 12 et le projet sera complété.

- Il y a un surplus de budget et un gain de temps équivalent à cinq sprints en avance (voir Figure 7.1 ci-bas).

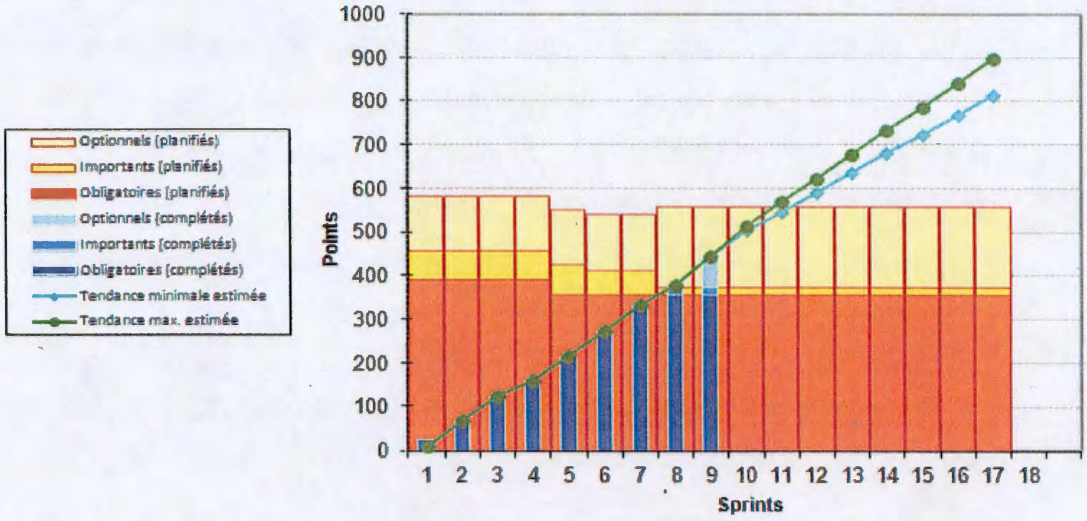


Figure 7.1 Statut du projet

Ces résultats confirment que les méthodes de développements utilisées ont été adéquates. À noter qu'un grand gain de temps est dû à l'utilisation de l'outil *Graphviz* pour la génération du graphe de dépendances entre les processus, l'envoi des messages *SMS* par le biais de courriel et l'utilisation de *Liferay,* qui comporte plusieurs librairies prêtes à utiliser (gestion des usagers et sécurité, création du *wiki,* enregistrement des fichiers, exécution des tâches selon un intervalle prédéfini).

7.2 Évaluation du produit

Comparant l'état du logiciel actuel avec ce qui était planifié, on constate que le produit répond adéquatement aux besoins pour qu'il soit déployé et utilisé (toutes les fonctionnalités critiques et importantes ont été réalisées). Cette comparaison était progressive et faite à chaque fin de sprint (rétrospective et revue).

La vérification était plus simple que d'habitude parce que c'est moi qui est à la fois l'utilisateur et le développeur. Pour être plus spécifique sur comment les validations

ont été effectuées, chaque récit utilisateur comporte des conditions de satisfaction qui ont été définies au départ et ont été vérifiées après le développement.

Ce qui reste à accomplir ou à améliorer touche la documentation et les tests automatisés pour aider les utilisateurs et aider les futurs développeurs à maintenir le logiciel. Quelques améliorations sur la gestion des *flags* (support de *flags* multiples) et la gestion de la configuration (temps de réinitialisation de la vue hebdomadaire des processus qui est fixe pour le moment) pourraient être faites pour rendre le logiciel plus générique.

Pour être plus précis quant à l'évaluation du produit, il faut vérifier et valider le logiciel.

Pour valider un logiciel, il faut s'assurer qu'il répond aux critères d'acceptations qui ont été définis lors de la rédaction des récits. C'est donc l'utilisateur qui doit confirmer si le logiciel et ses fonctionnalités répondent bien aux besoins.

À la fin du neuvième sprint, le logiciel avec les fonctionnalités actuelles répondait à plus que ce qui était attendu. Le logiciel est capable de gérer et surveiller les processus en temps réel, ainsi leurs dépendances (serveurs, gestion de chemins, cédule des processus, jours fériés, contacts personnels, fichiers *flags* et interdépendances). Un écran principal et flexible a été mis en place dans ce but. Les statuts des processus se mettent à jour régulièrement sans aucune interaction de la part de l'utilisateur. De plus, l'utilisateur peut se référer à un *wiki* pour chercher la documentation spécifique à un processus. Il peut aussi lister les personnes à contacter lorsque des problèmes surviennent. Les problèmes peuvent être enregistrés dans le système pour une consultation future.

Finalement, deux méthodes de notifications de statuts des processus ont été mises en place, soit la notification par courriel et la notification par des messages textes (SMS). Voir l'appendice B pour plus de détails sur les fonctionnalités du logiciel actuel.

Pour vérifier un logiciel, c'est le développeur qui doit s'assurer que le logiciel est conçu de la bonne manière. Pour simplifier cette tâche, il est utile de mettre en place des outils qui aident à faire cette vérification.

Dans le cas de ce projet, SonarQube a été mis en place pour vérifier la qualité du logiciel. SonarQube vérifie plusieurs aspects de la qualité et calcule plusieurs métriques (complexité, pourcentage de commentaires et documentation, bogues et leurs sévérités, duplication de code, *code smells*, anomalies de conception, vulnérabilités de sécurité, pourcentage du code couvert par des tests, degré de la maintenabilité, etc.).

La figure (Figure 7.2) ci-bas montre l'ensemble des métriques générées par SonarQube à la fin du neuvième sprint (les métriques d'acceptation sont configurables, mais j'ai conservé celles de SonarQube) :

Figure 7.2 Métriques de qualité produites par SonarQube

La valeur de l'indicateur nommée *Quality Gate* est une indication synthétique pour savoir si le logiciel engendre des problèmes. Les valeurs *Passed*, *A* et zéro signifient que le logiciel est dans un bon état.

Des critères additionnels doivent être vérifiés, mais ces vérifications ne peuvent pas être faites par les outils. Les critères architecturaux, la portabilité, l'évolutivité du logiciel et la documentation technique sont de même des métriques importantes à considérer.

Le logiciel a été conçu de façon à ce qu'il soit extensible, générique, configurable, portable, facile à modifier et maintenir. Ceci a été possible grâce à l'architecture choisie et aux patrons de conception utilisés lors du développement (voir chapitre 5).

CONCLUSION

Plusieurs leçons peuvent être tirées de notre expérience dans la réalisation de ce projet :

- Les méthodes agiles donnent plus de visibilité sur l'avancement du projet. Les problèmes peuvent être détectés et rapportés plus tôt afin d'appliquer les ajustements nécessaires avec moins de perte de temps et de coût et éviter l'accumulation de dettes techniques. Pour être plus précis, ce sont les pratiques suivantes qui ont permis d'avoir ce succès : développement avec des sprints itératifs, revues de sprints et rétrospectives faites à la fin de chaque sprint, utilisation d'indicateurs de suivi et le graphe de Sunset riche en informations.
- Pour une première utilisation de la méthode d'estimation basée sur les indices de complexité, j'ai trouvé que cette méthode était fiable et sûre.
- Il est bien connu qu'avec les méthodes agiles, il ne faut pas trop détailler les fonctionnalités et les récits utilisateurs. Ainsi, la décomposition des récits dans ce projet a été faite en suivant la règle INVEST (voir page 13). La décomposition était plus procédurale et traitait plus des aspects liés aux besoins d'affaire que des aspects techniques. L'exploration des aspects techniques, comme les outils et les librairies, peut révéler plus de détails sur les tâches et cela pourrait réduire leur niveau de complexité et le temps estimé pour les accomplir. Par exemple, la tâche comprenant la notification par des messages textes (*SMS*) était évaluée à un niveau complexe et d'une durée d'un sprint pour être accomplie, tandis qu'il existe deux techniques (envoi de messages SMS en utilisant le service fourni par une compagnie de

télécommunication qui permet d'envoyer des SMS via un courriel ou utilisation d'un service fourni par une compagnie tierce qui permet d'utiliser une librairie prête à être utiliser afin d'envoyer des messages SMS) qui permettent de l'accomplir en quelques heures.

Finalement, la stratégie adoptée dans ce projet nous a semblé optimale et présentait de nombreux avantages en la comparant avec d'autres stratégies que j'avais utilisées sur des projets antérieurs, où le développement était basé sur le cycle en cascade et des estimations basées sur l'expérience personnelle sans aucune règle spécifique. Il est trop tôt pour affirmer que cette méthode est meilleure que les autres. L'important est de conserver les résultats obtenus et de réutiliser cette méthode dans des projets futurs, afin d'établir des indices de comparaisons clairs et améliorer les bénéfices apportés par l'utilisation de cette approche.

APPENDICE A

DÉTAILS D'AVANCEMENT D'UN SPRINT À UN AUTRE

Le tableau ci-bas fournit un résumé des événements cruciaux survenus durant chaque sprint.

Numéro Sprint	**(-) Problèmes rencontrés** **(+) Amélioration à faire**	**Améliorations faites** **Commentaires**
Sprint 1	- Le temps requis était deux fois plus que le temps planifié. Dans le premier sprint, il fallait identifier la bonne architecture et la mettre en place. + Allouer plus de temps pour la complexité inattendue, la revision de code, le déploiement, la documentation, les tests, la couche de service de la BD. + Une refactorisation de la façon dont les données ont été saisies pour ajouter un processus était	

	nécessaire.	
Sprint 2	+ Comme le code était en amélioration et modification continue, c'était mieux de retarder les tests, la documentation et les validations au dernier moment possible.	+ La planification et l'accomplissement de ce sprint ont été cohérents (le temps consommé était égal au temps estimé). + Les améliorations élaborées dans le premier sprint ont été prises en considération.
Sprint 3	+ Il fallait réviser le contexte pour faciliter les tâches futures (beaucoup de cohésion).	+ Les tâches ont été accomplies plus rapidement que d'habitude, surtout que les implémentations étaient semblables à des tâches accomplies auparavant.
Sprint 4	- Un problème technique avec *Jenkins* est survenu suite à la mise à jour du *BitBucket*. - La génération d'un schéma de dépendances entre les processus n'a pas été exploitée adéquatement au début du projet (les librairies de *Graphviz* et de *Liferay* ont sauvé la situation).	+ Un formateur de code a été mis en place. + Les librairies de *Graphviz* et de *Liferay* ont permis de résoudre la complexité de la tâche de génération d'un schéma de dépendances entre les processus.
Sprint 5	- Un problème avec *Liferay* a été rencontré (écrasement de l'attribut *Name* des composants HTML,	+ La librairie *log4j* a été ajoutée pour le besoin du *logging*.

	qui ajoute le nom du fragment visuel comme préfixe). - Un problème a été rencontré avec l'intégration du *JQuery*. + Une dette technique elevée de la qualité du code était détectée par SonarQube.	+ Le plug-in *JAutoDoc* a été ajouté pour générer automatiquement les *Javadoc*. + L'outil SonarQube a été ajouté pour assurer la qualité du code.
Sprint 6	- Quelques difficultés avec l'affichage et l'utilisation des cases à cocher en relation avec *Liferay* ont été rencontrées. + Une révision du carnet de produits était nécessaire.	+ Carnet de produits était révisé et plusieurs points de fonction ont été éliminées.
Sprint 7	- Un problème a été rencontré avec la détection des zones de temps pour faire les comparaisons.	+ L'aspect multilangage a été ajouté. + L'outil *Quartz Job scheduler* a été mis en place pour exécuter des tâches à un intervalle prédéfini ou à un moment spécifique. + L'outil *EhCache* comme gestionnaire de *cache* a été mis en place pour améliorer la performance. + L'utilisation des options avancées du *Datatable* a été mise en place pour

		faire la mise à jour d'une cellule dans la table.
Sprint 8		+ L'ajout de la notification des événements avec courriel et *SMS* a été mis en place. + La création du *wiki* a été effectuée. + Des ajustements de tâches dans le carnet de produit (tests, documentation et revue de code) ont été faits.
Sprint 9		+ Une amélioration au niveau de la vérification des *flags* a été faite. La vérification est dans une classe *Thread* séparée au lieu qu'elle soit attachée au *Thread* principal afin d'améliorer la performance. + Tous les problèmes signalés par l'outil SonarQube ont été résolus.

Tableau A.1 Avancement d'un sprint à un autre

APPENDICE B

PRÉSENTATION DÉTAILLÉE DES FONCTIONNALITÉS DU LOGICIEL CWA

Cet appendice comporte des figures présentant l'ensemble des fonctionnalités du logiciel.

- Menu principal :

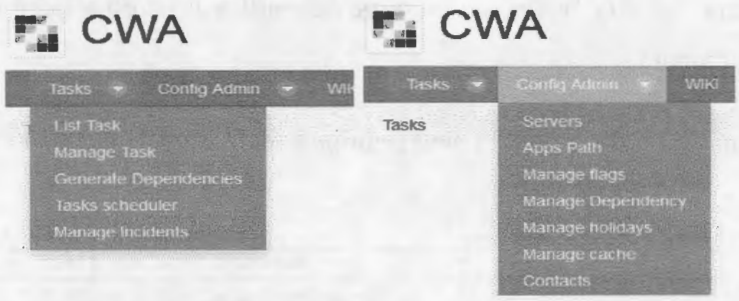
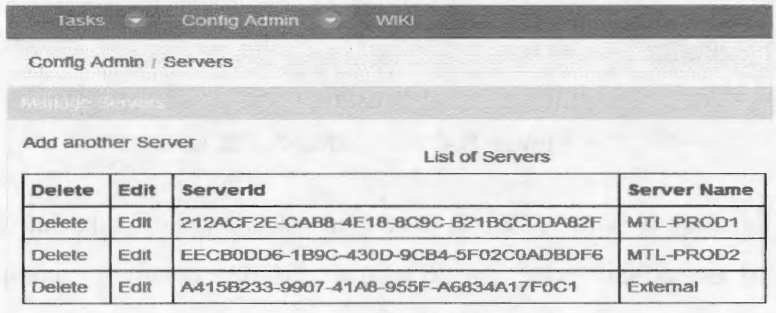


Figure B.1 Menu principal du logiciel

- Gestion des serveurs :



Delete	Edit	ServerId	Server Name
Delete	Edit	212ACF2E-CAB8-4E18-8C9C-B21BCCDDA82F	MTL-PROD1
Delete	Edit	EECB0DD6-1B9C-430D-9CB4-5F02C0ADBDF6	MTL-PROD2
Delete	Edit	A415B233-9907-41A8-955F-A6834A17F0C1	External

Figure B.2 Gestion / listage des serveurs

Si on clique sur *Add another server*, le bouton nous dirige vers la page de création d'un serveur :

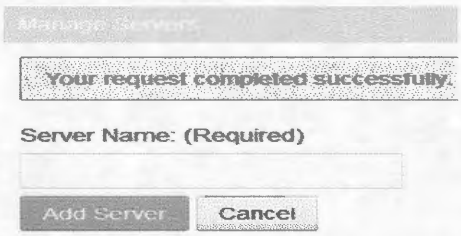


Figure B.3 Création d'un serveur

Si on clique sur *Cancel*, la page nous dirige vers la liste des serveurs.

Si on clique sur *Add Server*, un message de confirmation est affiché avec les détails du serveur ajouté.

Si on clique sur *Edit*, le bouton nous redirige vers la page de l'édition d'un serveur :

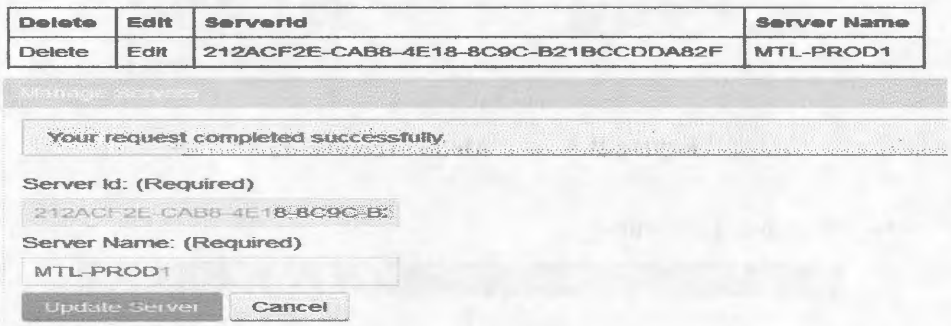


Figure B.4 Édition d'un serveur

Si on clique sur le bouton *Delete*, le serveur sélectionné est supprimé et un message informatif est affiché pour nous confirmer la suppression. Le comportement des boutons est semblable pour toutes les fonctionnalités. Pour les prochaines fonctionnalités, ces détails vont être omis.

- Gestion de chemins (*Apps Path*) :

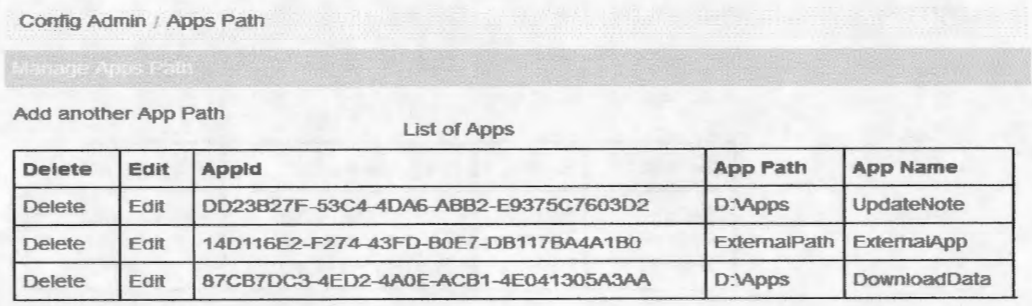


Figure B.5 Gestion / listage des chemins

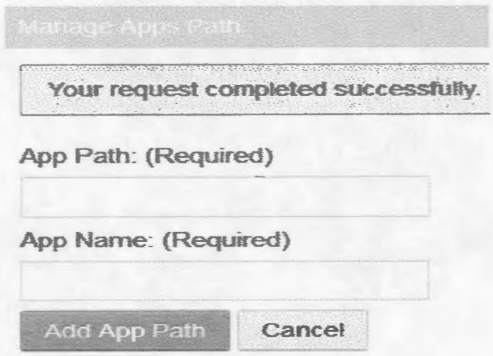


Figure B.6 Ajout d'un chemin

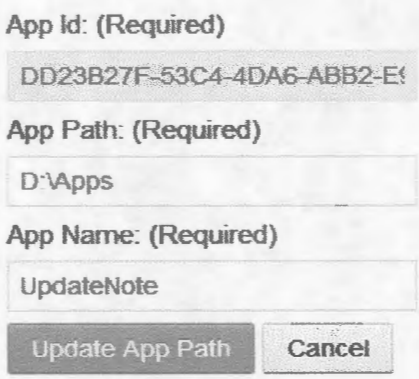


Figure B.7 Édition d'un chemin

- Gestion des fichiers *flags* :

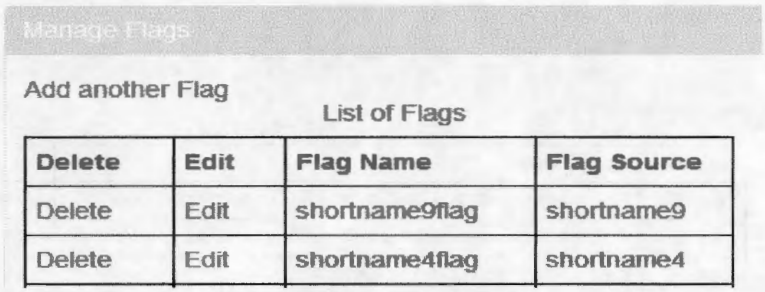


Figure B.8 Gestion / listage des fichiers *flags*

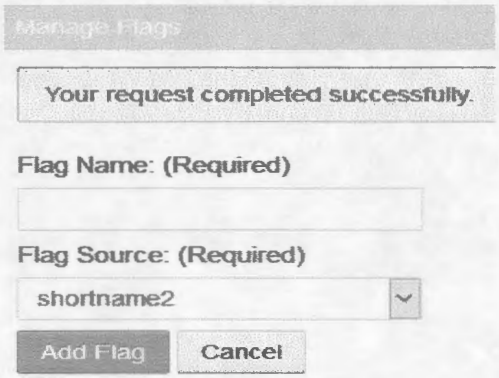


Figure B.9 Création d'un fichier *flag*

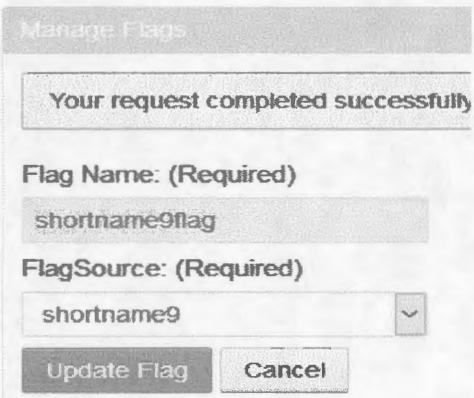


Figure B.10 Édition d'un fichier *flag*

- Gestion de dépendances :

Manage Dependency

Add another dependency

Delete	Edit	Task Name	Flag Name	Flag Source	DependencyDirection
Delete	Edit	shortname2	shortname1flag	shortname1	Input
Delete	Edit	shortname7	shortname6flag	shortname6	Input

Figure B.11 Gestion / listage des dépendances

Manage Dependency

TaskName: (Required)

shortname2

FlagName: (Required)

shortname9flag (shortname9)

DependencyDirection: (Required)

Input

Add Dependency | Cancel

Figure B.12 Création d'une dépendance

Manage Dependency

Dependency Id: (Required)

48596245-751A-47A8-89FC-B5E

TaskId: (Required)

shortname2

FlagName: (Required)

shortname1flag (shortname1)

DependencyDirection: (Required)

Input

Update Dependency | Cancel

Figure B.13 Édition d'une dépendance

- Gestion des jours fériés :

Figure B.14 Gestion / listage des jours fériés

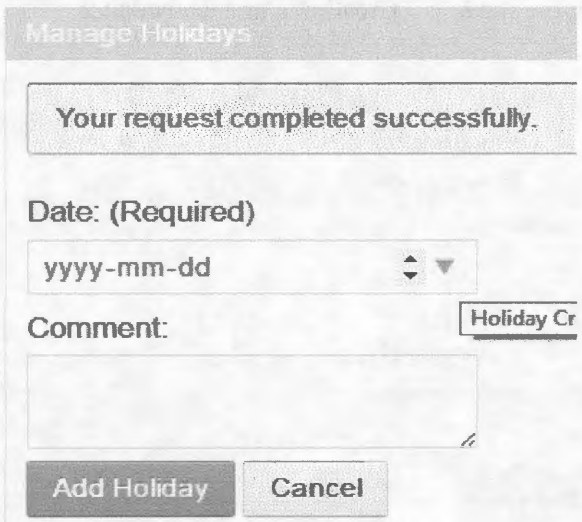


Figure B.15 Ajout d'un jour férié

- Gestion de contacts :

Figure B.16 Gestion / listage des contacts

Figure B.17 Création d'un contact

Figure B.18 Édition d'un contact

- Gestion du *cache* :

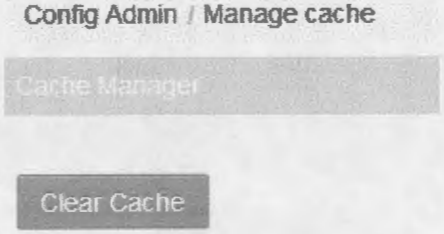


Figure B.19 Nettoyage de la *cache*

- Gestion de tâches (processus) :

Add another Task

Delete	Edit	Short Name (Click for details)	Server Name	Start Time	End Time	Email Subject	StartUpApplicationPath	Macro\ Sub	Comment
Delete	Edit	shortname2	MTL-PROD2	17:00:00	17:15:00	sendFile	D:\Apps\UpdateNote	sub2	send file to
Delete	Edit	shortname1	MTL-PROD1	16:00:00	16:30:00	subject	D:\Apps\DownloadData	sub1	

Figure B.20 Gestion / listage des tâches

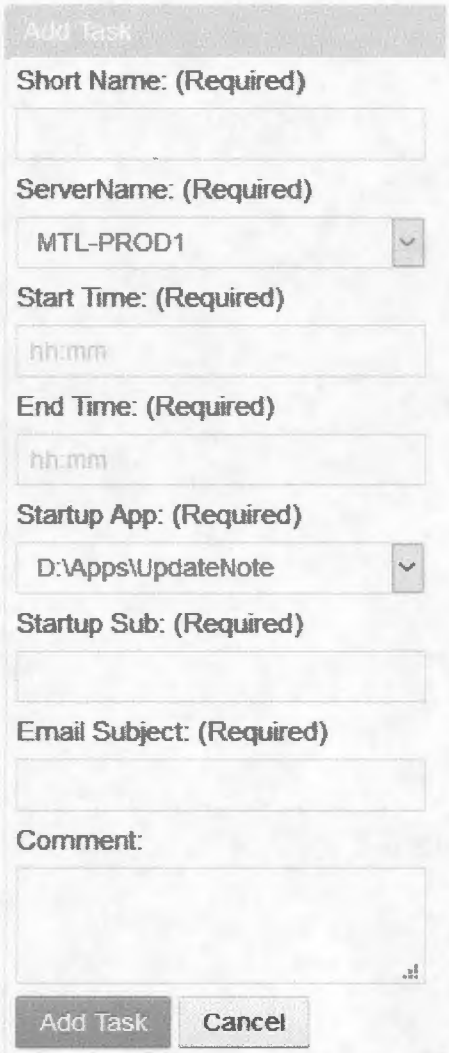


Figure B.21 Création d'une tâche

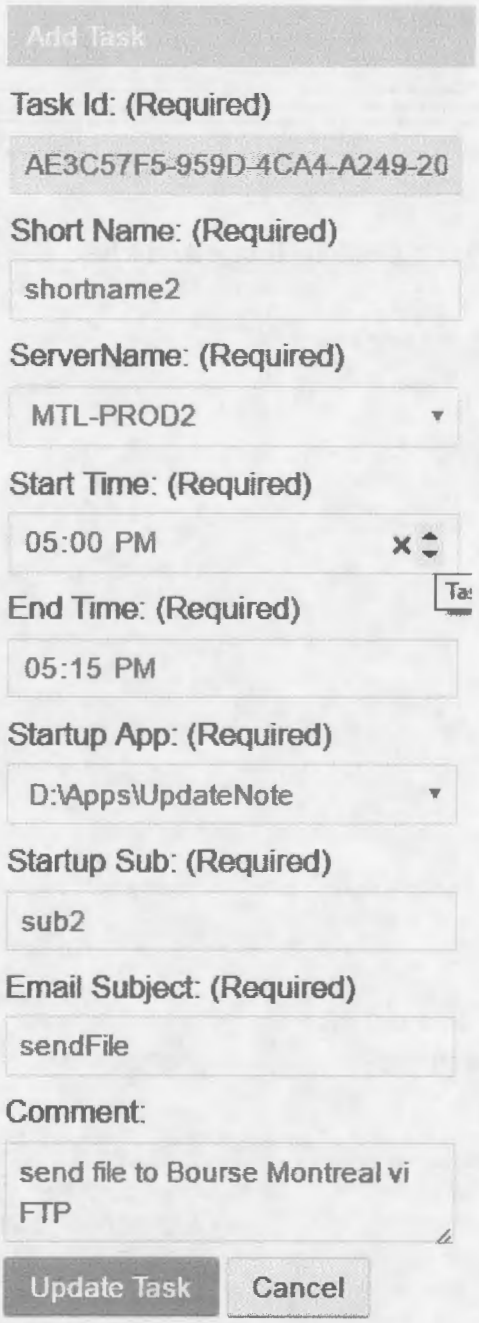


Figure B.22 Édition d'une tâche

Si on clique sur le nom d'une tâche, on est dirigé vers une page qui affiche tous les détails de cette tâche. Cet écran permet d'affecter ou de changer un contact ou une dépendance pour une tâche.

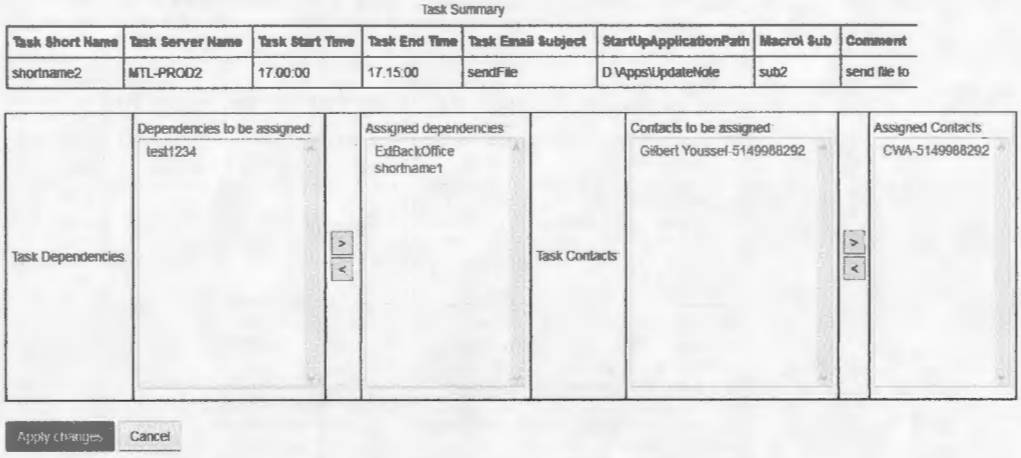


Figure B.23 Affichage des détails d'une tâche

- Génération du schéma de dépendances :

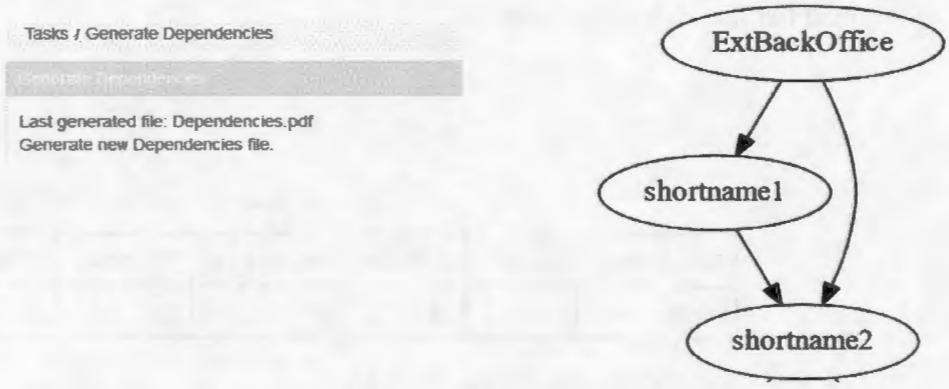


Figure B.24 Génération du schéma de dépendances

On a le choix d'accéder au dernier schéma généré ou de générer un autre schéma.

- Gestion des cédules pour les tâches :

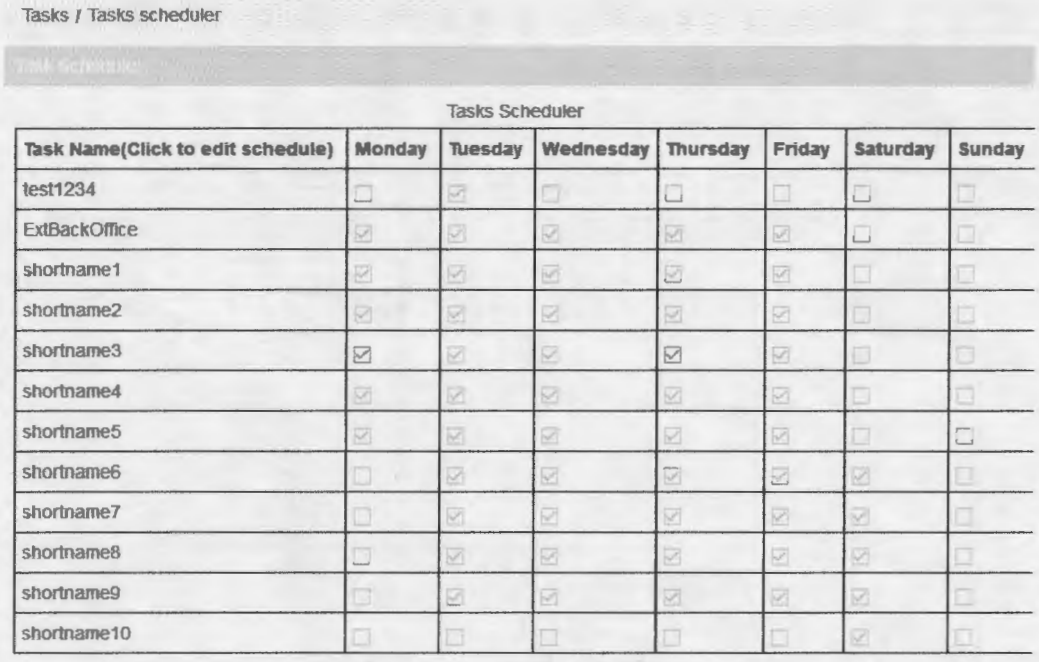


Figure B.25 Gestion / listage des cédules des tâches

Si on clique sur le nom d'une tâche, on est dirigé vers une page qui permet de modifier la cédule d'une tâche :

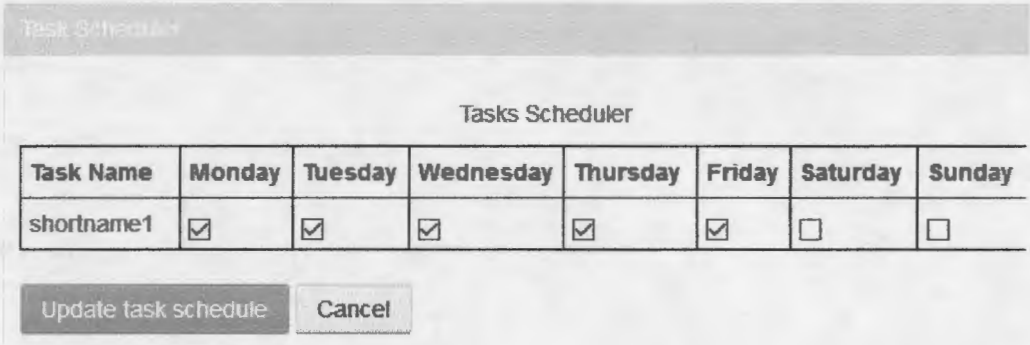


Figure B.26 Modification de la cédule d'une tâche

- Gestion des incidents :

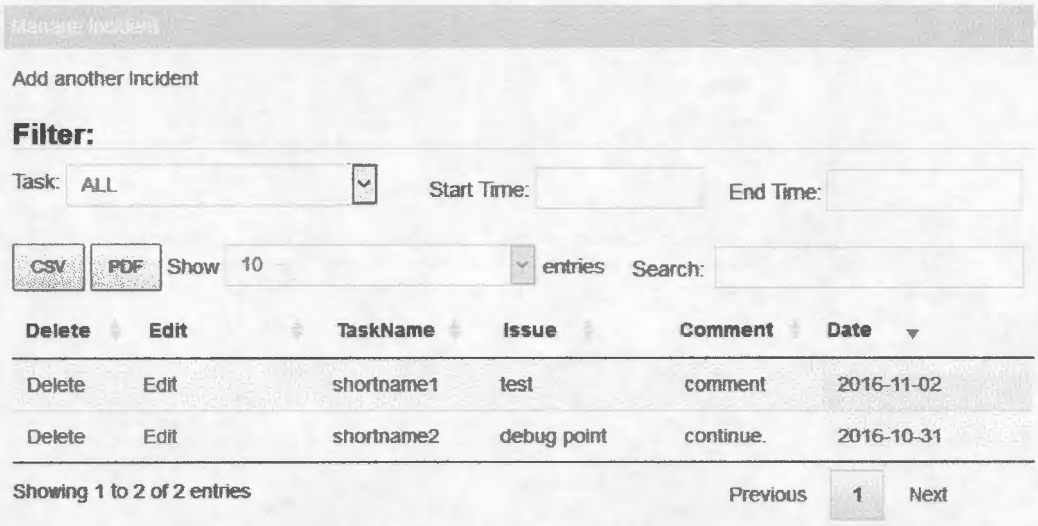


Figure B.27 Gestion / listage des incidents

Les deux boutons (PDF et CSV) permettent de générer un fichier qui comporte une liste des incidents filtrés au besoin.

Figure B.28 Ajout / édition d'un incident

- Surveillance des tâches :

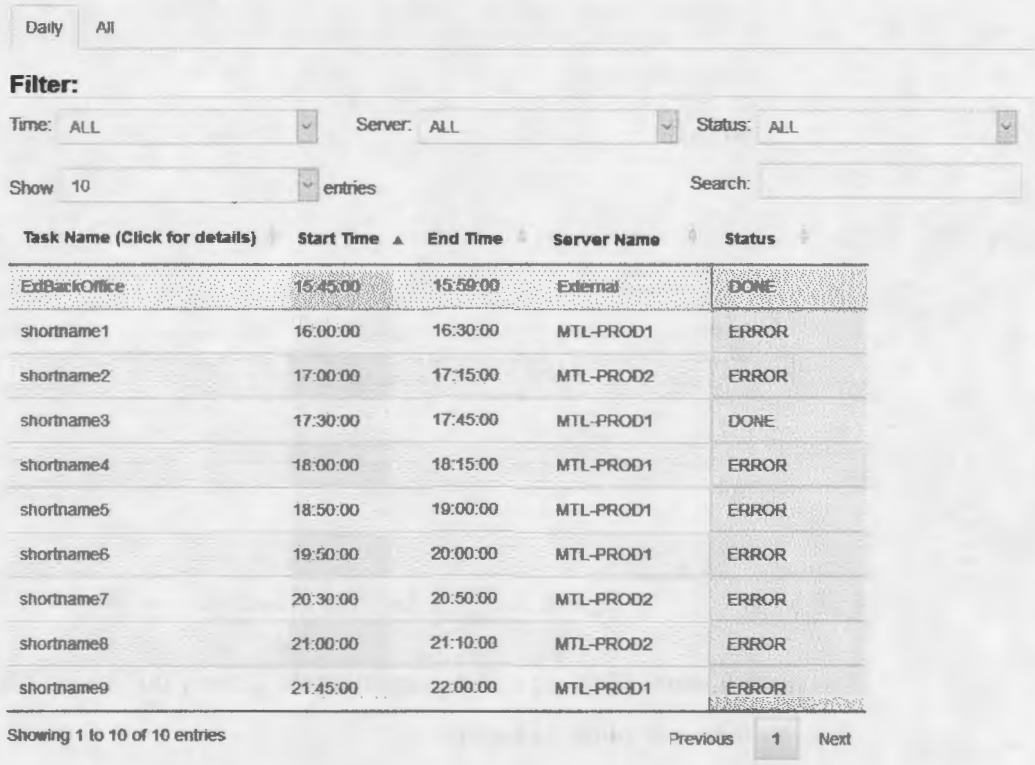


Figure B.29 Surveillance des tâches

Par défaut, cet écran affiche les tâches qui sont cédulées pour la journée.

De plus, ceci donne la possibilité de voir toutes les tâches, les détails d'une tâche et de filtrer les tâches par serveur, temps, statut ou nom.

Les statuts des tâches sont mis à jour régulièrement.

Le système vérifie l'existence du fichier *flag* pour chaque tâche, si le fichier *flag* n'est pas reçu au moment où il est censé exister (temps actuel plus grand que le temps de fin de la tâche), le statut se change en « erreur » sur l'écran et des messages de notifications sont envoyés par courriel et SMS.

- Gestion du *wiki* :

  La gestion des pages de contenu Web pour le *wiki* est faite par le gestionnaire de contenu Web de Liferay :

Figure B.30 Gestionnaire de contenu Web

Une page de recherche a été créée pour permettre de faire des recherches dans le *wiki*:

Figure B.31 Recherche de contenu Web dans le *wiki*

BIBLIOGRAPHIE

Ambler, S. (2002). *Agile Modeling.* New York: John Wiley & Sons, Inc.

Wake, B. (2003, 08 17). *INVEST in Good Stories, and SMART Tasks.* Consulté le 11 janvier 2016, sur Exploring Extreme Programming: https://www.agilealliance.org/glossary/invest/

Schwaber, K. (2004). *Agile Project Management With Scrum.* Redmond, WA: Microsoft Press.

Cohen, M. (2004). *User story applied.* Boston: Addison-Wesley Professional.

Cohen, M. (2006). *Agile Estimating and Planning.* Boston, MA: Pearson Education, Inc.

Kniberg, H. (2007). Scrum et XP depuis les Tranchées. C4Media Inc.

Standish Group International, Inc. (2013). *CHAOS MANIFESTO.* The Standish group.

Sanders, R. (2014, Janvier 02). *A simple project effort estimation utility*. Consulté le 10 janvier 2016, sur http://sanderstechnology.com/2014/a-simple-project-effort-estimation-utility/12854/#.WFigKfkrKUl

Pyxis. (2015, Septembre 20). Sunset Graph. Montréal, Québec, Canada.

Divers document utilisés dans les cours de la maîtrise en génie logiciel, surtout les suivants :

- Trudel, S. (2014). Gestion de projet en génie logiciel logiciels : notes de cours et illustrations, MGL7315. Université de Québec à Montréal, Départment d'informatique.
- Seguin, N. (2014).Vérification et assurance qualité de logiciels : notes de cours et illustrations, MGL7560. Université de Québec à Montréal, Départment d'informatique.
- Trudel, S. (2014). Processus de développement agile : notes de cours et illustrations, MGL7250. Université de Québec à Montréal, Départment d'informatique.
- Moha, N. (2015). Qualité et productivité des outils logiciels : notes de cours et illustrations, MGL7760. Université de Québec à Montréal, Départment d'informatique.
- Ross, Y. (2015). Principes et applications de la conception de logiciels : notes de cours et illustrations, MGL802. École de technologie supérieure, Département de génie logiciel.
- April, A. (2015). Réalisation et maintenance des logiciels : notes de cours et illustrations MGL804. École de technologie supérieure, Département de génie logiciel.