RAPPORT FINAL D'ALTERNANCE

DÉVELOPPEMENT D'UN EXTRANET CLIENT PAR L'INTERMÉDIAIRE D'UNE APPLICATION DÉDIÉE À LA GESTION DE CLIENTS ET D'UTILISATEURS

RÉALISÉ PAR

Thomas CHARMES

POUR L'OBTENTION DE LA LICENCE PROFESSIONNELLE API-DAE

SOUS LA DIRECTION DE

M. Fabien MICHEL

ANNÉE UNIVERSITAIRE 2019-2020

Remerciements

Je tiens en premier lieu à remercier chaleureusement les cofondateurs de l'entreprise "Do&Go", Messieurs Pierre PONTFORT et Matthieu DE PUYSEGUR, qui n'ont pas hésité à prendre le pari de donner sa chance d'entrer dans le monde de l'entreprise à un jeune étudiant en m'intégrant pleinement à leur équipe. Je les remercie également pour avoir consacré, sans compter, leur temps à ma formation, ce qui a grandement contribué à ma montée en compétences progressive tout au long de cette année malgré les conditions de travail inédites de ces derniers mois.

Je remercie également Monsieur Fabien MICHEL, pour m'avoir enseigné des connaissances que j'utilise chaque jour en entreprise, concernant le langage Java et Java pour le web. Je le remercie également pour avoir suivi avec attention l'évolution de la situation de mon alternance, notamment durant cette période de confinement au cours de laquelle chacun a dû s'adapter.

Je remercie l'ensemble du corps enseignant de la licence professionnelle API-DAE de l'IUT de Montpellier pour leur disponibilité tout au long de l'année ainsi que pour leur aide lors de la réalisation de ce rapport de projet.

Enfin, je remercie tout particulièrement Monsieur Xavier PALLEJA, qui a été d'une grande réactivité pour répondre à toutes mes questions et me fournir des pistes pendant ma recherche d'entreprise, ainsi que Madame Justine DELEBARRE pour ses précieux conseils d'élaboration du rapport et de présentation orale.

Table des matières

Table des figures

Glossaire

Affinité conceptuelle, *voir aussi Règle de décroissance* : Terme utilisé par les développeurs lorsque les concepts informatiques étroitement liés d'un programme sont regroupés physiquement, c'est-à-dire qu'ils se situent « géographiquement » proches les uns des autres. Cela concerne notamment les méthodes interdépendantes au sein d'une classe.

Bean : Objet Java représenté par une classe publique, comportant un constructeur par défaut public et sans paramètre. Cette classe ne contient pas d'attributs publiques mais des propriétés privées accessibles grâce à des méthodes publiques : les accesseurs en lecture et en écriture.

Camelcase : Style d'écriture consistant à remplacer les espaces par la majuscule du mot suivant. Exemple : exempleDeCamelcase.

Catastrophe ferroviaire : Terme informatique employé par les développeurs lorsque de trop nombreuses méthodes sont appelées les unes à la suite des autres sans que les valeurs renvoyées soient stockées dans des variables avant d'être manipulées. Cela peut provoquer des bogues et une lisibilité/maintenabilité du code plus complexe.

Extranet : Intranet dont l'accès est étendu à certaines personnes extérieures, le plus souvent des fournisseurs, clients ou partenaires. Par opposition, un réseau intranet, se limite au réseau interne à l'organisation, sans utiliser d'infrastructure tierce.

Fabrique abstraite : Patron de conception utilisé en génie logiciel orienté objet. Elle fournit une interface pour créer des familles d'objets liés ou interdépendants sans avoir à préciser au moment de leur création la classe concrète à utiliser.

JBoss : Serveur d'application Java EE écrit en Java et utilisé par Moovapps.

Loi de Demeter : Loi stipulant qu'un module ne doit pas connaître les détails d'un objet interne qu'il manipule. Cette loi, également appelée « principe de connaissance minimale » et inventée en 1987, peut être résumée par la phrase « Ne parlez qu'à vos amis immédiats ».

Méthodes CRUD : Désigne les quatre opérations de base (« Create », « Read », « Update », et « Delete ») destinées à gérer la persistance des données, en particulier le stockage d'informations en base de données.

Moovapps : Solution accompagnant les entreprises dans leur transformation numérique.

Moovapps Studio : Environnement de création des applications Moovapps.

Moovapps Workplace : Environnement de connexion d'un utilisateur Moovapps.

POJO : En anglais « Plain-Old Java Object », bon vieil objet Java. Classe Java indépendante du reste du programme qui permet d'assurer la propreté du système et de faciliter les tests en dehors de l'environnement de déploiement de l'application, notamment les tests de concurrence lors de la gestion de l'aspect « multi-thread ». Néanmoins, ce sujet n'est pas abordé dans ce rapport.

Principe de moindre surprise, *voir aussi Principe de Ward* : Principe décrivant que toute méthode ou classe doit implémenter les comportements auxquels un autre programmeur peut raisonnablement s'attendre. Cela implique notamment que cette dernière doit être correctement nommée.

Principe de responsabilité unique, *voir aussi Principes SOLID* : Principe faisant partie des bonnes pratiques de programmation décrivant qu'une méthode ne doit avoir qu'une seule responsabilité. C'est-à-dire qu'elle ne doit avoir qu'une seule fonction.

Principe de ségrégation d'interfaces, *voir aussi Principes SOLID* : Principe faisant partie des bonnes pratiques de programmation incitant à préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale.

Principe de substitution de Liskov, *voir aussi Principes SOLID* : Principe faisant partie des bonnes pratiques de programmation décrivant qu'une instance de type T doit pouvoir être remplacée par une instance de type G, tel que G sous-type de T, sans que cela ne modifie la cohérence du programme.

Principe de Ward, *voir aussi Principe de moindre surprise* : Principe faisant partie des bonnes pratiques de programmation décrivant que le code est propre lorsque chaque méthode que le lit lecteur correspond presque parfaitement à ce à quoi il s'attendait.

Principe d'inversion des dépendances, *voir aussi Principes SOLID* : Principe faisant partie des bonnes pratiques de programmation indiquant qu'un objet doit dépendre d'abstractions, et non pas d'implémentations.

Principe ouvert/fermé, *voir aussi Principes SOLID* : Principe faisant partie des bonnes pratiques de programmation décrivant qu'un programme doit pouvoir être ouvert à d'éventuelles extensions tout en restant fermé aux modifications des classes qui fonctionnent. En effet, cela pourrait détériorer la qualité du code, voire modifier le comportement du programme.

Principes SOLID : Principes de programmation fondamentaux à respecter pour obtenir un code propre. Ces principes sont les suivants et sont définis plus en détails individuellement dans ce glossaire : ***principe de responsabilité unique***, ***principe ouvert/fermé***, ***principe de substitution de Liskov***, ***principe de ségrégation d'interfaces*** et ***principe d'inversion des dépendances***.

Principes STUPID : Mauvaises pratiques de programmation à éviter : l'utilisation du singleton, le fort couplage, la non-testabilité du code, l'optimisation prématurée, l'utilisation de noms inappropriés ainsi que la duplication de code.

Regex : Expression rationnelle, appelée parfois « motif ». Décrit un ensemble de caractères possibles à l'aide d'une syntaxe spécifique. L'application d'une expression rationnelle sur une chaîne de caractères initiale permet d'en extraire les caractères qui respectent l'expression.

Règle de décroissance, *voir aussi Affinité conceptuelle* : Règle décrivant que le code doit être pouvoir lu de haut en bas, comme un livre, en respectant donc l'affinité conceptuelle des éléments.

Refactoring : En français « ré-usinage de code ». Opération consistant à retravailler le code source d'un programme informatique de manière à ce qu'il soit plus lisible, maintenable, et générique. Cette opération ne modifie pas le comportement initial du programme.

Introduction

Aujourd'hui, les nouvelles technologies prennent de plus en plus d'ampleur dans notre vie quotidienne, nous offrant ainsi des solutions pratiques adaptées à nos besoins. Un grand nombre d'entreprises souhaite tendre vers le numérique à travers différentes étapes, comme la gestion électronique des documents (GED) ou la gestion des demandes internes.

Cette transition, bien que très pratique, nécessite toutefois une restructuration du système d'information et/ou du système informatique de l'entreprise mais cela comporte des contraintes techniques. L'objectif de l'entreprise « Do&Go » est d'accompagner ces entreprises au cours de leur transition numérique. Les deux enjeux majeurs de cette restructuration sont la rapidité de traitement des demandes internes au sein d'une entreprise ainsi que la préservation de l'environnement.

Le projet qui sera développé dans ce rapport consiste à mettre en place un **extranet** destiné aux clients. Pour cela, une application dédiée à la gestion de clients et d'utilisateurs externes sera développée. Ce rapport détaillera la partie de l'application consacrée à la création d'un nouvel utilisateur.

Après une mise au point dans le cahier des charges du contexte et de l'ensemble des besoins, le rapport technique présentera les choix de conception effectués pour développer cette application ainsi que le détail de certains points importants du code. Cette partie suivra également une chronologie représentant l'évolution du code au fil du temps. Les résultats, quant à eux, seront exposés dans une troisième partie. Enfin, la quatrième et ultime partie présentera les méthodes de travail habituelles de l'entreprise « Do&Go » ainsi que l'adaptation de ces méthodes au cours des derniers mois nécessaire face à la crise sanitaire du Covid-19.

I Présentation de l'entreprise

L'entreprise Do&Go, basée à Castries dans l'Hérault, est intégratrice de la solution **Moovapps**. Cette dernière a été éditée par la société Visiativ dans le but de répondre aux besoins spécifiques des entreprises en réalisant des applications sur mesure.

Fondée en juillet 2019, l'entreprise se compose de quatre employés. Pierre PONTFORT est développeur, il réalise des développements spécifiques en java, des scripts en javascript pour répondre aux besoins spécifiques des clients ainsi que du déploiement et de la mise à jour d'applications. En effet, certaines demandes ne peuvent pas être réalisées uniquement en utilisant les applications Moovapps existantes. Matthieu DE PUYSEGUR est consultant fonctionnel, il gère les parties administratives de l'entreprise, recherche de nouveaux projets et assure des formations auprès des clients. Ce sont les cofondateurs de « Do&Go ». En janvier 2020, Pierrick MAILLARD a intégré l'équipe en tant que consultant fonctionnel.

L'accès au site internet de l'entreprise s'effectue via le lien *http://doandgo.fr/.*

2 Cahier des charges

Cette partie détaille l'expression des différents besoins et propose une solution pour y répondre.

2.1 Présentation du sujet

Le client « Sud Inox », basé à Nîmes, est une entreprise formée plus de 30 personnes dont l'activité principale est de concevoir et de fabriquer des équipements inox pour la cuisine professionnelle et la grande distribution. Cette entreprise a décidé d'opter pour une solution de gestion électronique de ces documents ainsi que de ces demandes internes. Cette solution est une application web qui se nomme Moovapps. Elle peut être définie par un support de création d'applications (appelées « apps ») qui peuvent être soit génériques, c'est-à-dire déjà existantes et téléchargeables par le client, soit spécifiques aux besoins du client, et qui sont dans ce cas développées par l'entreprise Do&Go. Un exemple d'application générique est « Activity », qui permet aux salariés de l'entreprise de saisir leurs temps d'activités.

Pour comprendre comment les salariés d'une entreprise utilisent concrètement Moovapps, il est nécessaire de savoir comment s'organise l'arborescence de cette solution. Le schéma simplifié ci-dessous illustre son mécanisme global :

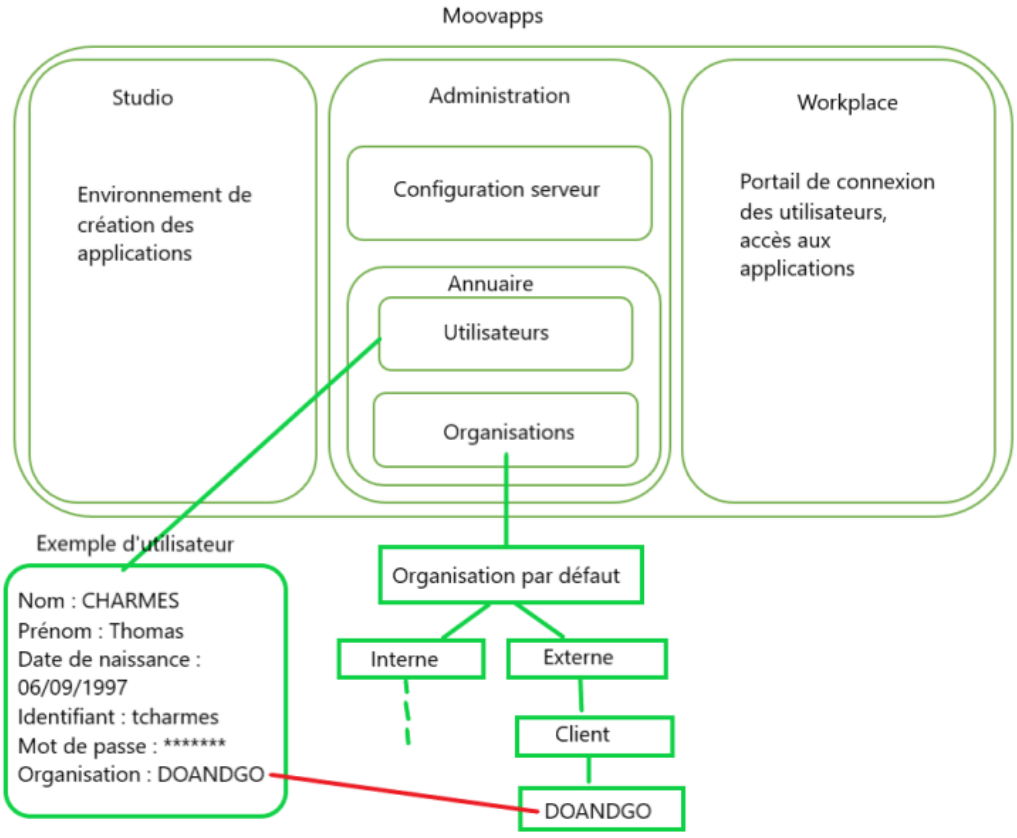

Figure 1 : Schéma simplifié de l'architecture de Moovapps

Lorsqu'un client, ici « Sud Inox », se procure une licence Moovapps, cette dernière est valable pour un certain nombre d'apps, pendant une certaine durée, et pour un certain nombre d'utilisateurs. Lors de l'installation de Moovapps, plusieurs catégories de services sont créées : le **studio**, l'administration et « **workplace** ». Le studio est l'environnement de création des applications et n'est accessible que par l'administrateur système (le « sysadmin »). « Workplace » est l'interface de connexion d'un utilisateur à partir duquel ce dernier accède aux applications créées. Enfin, l'administration est l'environnement de configuration. Ce dernier possède plusieurs parties telles que la configuration du serveur ou encore l'annuaire. C'est dans cet annuaire que l'administrateur système peut gérer les organisations et les utilisateurs. Un utilisateur possède des informations standards telles qu'un nom, un prénom, ou encore une adresse. De plus, chaque utilisateur est obligatoirement rattaché à une organisation. Un client est une organisation qui, lorsqu'elle n'est pas principale, est obligatoirement une sous-organisation de l'organisation « interne » ou de l'organisation « externe » (voir figure 1).

2.2 Analyse du contexte et description des besoins

Le nombre d'utilisateurs autorisés par la licence peut être réparti entre les utilisateurs internes et externes. Le client « Sud Inox » souhaite contrôler la création et la gestion des utilisateurs provenant de clients externes. Pour cela, l'application « Gestion des utilisateurs externes » a été développée dans le studio Moovapps. Celle-ci est composée de deux processus : « Gestion des clients » et « Gestion des utilisateurs ». Comme évoqué en introduction, c'est ce dernier processus qui sera détaillé dans ce rapport. L'application possède également un réservoir de données comportant une table de configuration. Cette table de configuration paramétrée par le gestionnaire contient les caractéristiques communes à tous les utilisateurs de création de l'identifiant et du mot de passe. Le fonctionnement global du processus « Gestion des utilisateurs » est récapitulé par le diagramme des cas d'utilisation suivant :

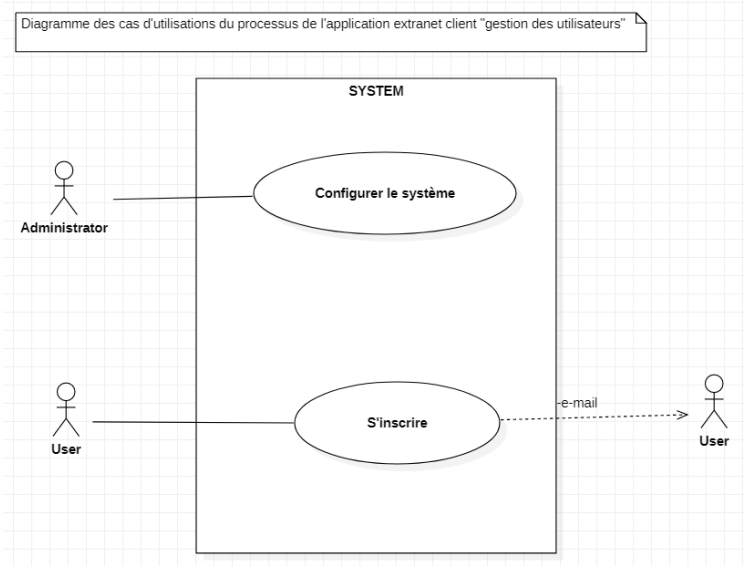

Figure 2 : Diagramme des cas d'utilisation du processus "Gestion des utilisateurs"

Le besoin principal du client est d'automatiser la gestion de l'identifiant et du mot de passe des utilisateurs externes. Pour cela, le gestionnaire peut choisir entre différentes options de création qui ont été créées dans la table de configuration.

Pour la génération de l'identifiant, le gestionnaire peut décider de se baser sur :

- Le nom et prénom de l'utilisateur
- L'email de l'utilisateur
- La société de l'utilisateur

Dans le cas où le gestionnaire opte pour la première option, l'identifiant sera égal à l'initiale du prénom suivi du nom de famille, sans accent et sans caractères spéciaux. Par exemple, si l'utilisateur se prénome Thomas CHARMES, son identifiant sera « tcharmes ». En cas de doublon, une valeur numérique sera ajoutée et incrémentée à chaque création d'utilisateur. Par exemple, si Tom CHARMES crée un compte après Thomas CHARMES, il lui sera attribué comme identifiant « tcharmes1 ».

Dans le cas où le gestionnaire opte pour la deuxième option, l'identifiant sera simplement l'adresse email de l'utilisateur.

Enfin, dans le cas où le gestionnaire opte pour la dernière option, l'identifiant sera égal à un préfixe commun à tous les utilisateurs, par exemple le nom de la société (paramétrable également dans la table de configuration) suivi d'un numéro unique. Par exemple : DOANDGO-0001. Il est également possible, dans ce cas là, de choisir l'index de départ (ici 1) ainsi que le nombre de caractères total du suffixe (ici 4). Cette valeur numérique est incrémentée à chaque création d'utilisateur.

Pour la génération du mot de passe, le gestionnaire peut décider d'opter pour :

- Un mot de passe simple
- Un mot de passe sécurisé

Dans le cas où le gestionnaire opte pour la première option, le mot de passe, s'il est généré aléatoirement, commencera obligatoirement par une majuscule et ne comportera pas de caractères spéciaux. Les seuls éléments paramétrables de ce mot de passe seront le nombre total de caractères ainsi que le nombre exact de chiffres qu'il contiendra. Un exemple de mot de passe simple pour une configuration de 8 caractères au total et de 2 chiffres peut être « Da2u3yre ».

Dans le cas où le gestionnaire opte pour la seconde option, le mot de passe, s'il est généré aléatoirement, pourra comporter des majuscules, des minuscules, des chiffres et des caractères spéciaux. Le nombre exact de chacun de ses éléments peut être indiqué par le gestionnaire dans la table de configuration. Un exemple de mot de passe sécurisé avec pour configuration 2 majuscules, 2 minuscules, 2 chiffres et 2 caractères spéciaux peut être « 1t7Höm@S ».

Le formulaire de configuration se présente comme suit :

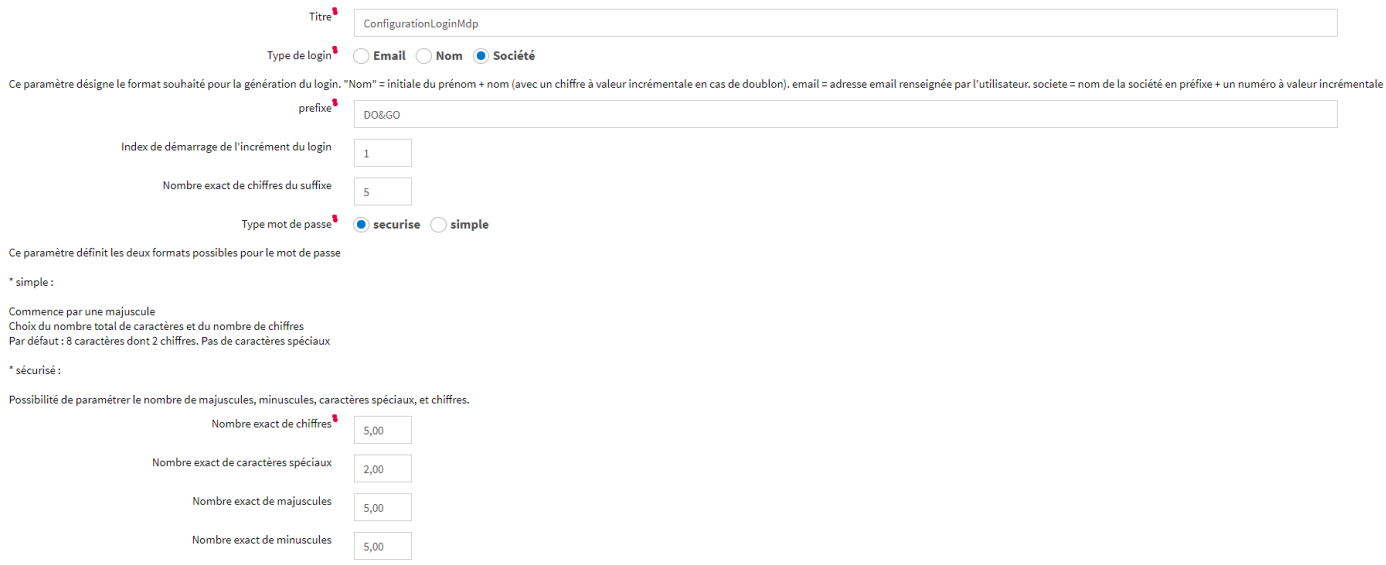

Figure 3 : Formulaire de configuration d'identifiant et de mot de passe

La table de configuration complétée possède cet aspect :

Titre	Type de login	Type mot de passe	prefixe	Nombre total de caractères	Nombre exact de chiffres	Nombre exact de majuscules
ConfigurationLoginMdp	Société	securise	DO&GO	8	5	5

Nombre exact de minuscules	Nombre exact de caractères spéciaux	Index de démarrage de l'incrément du login	Nombre exact de chiffres du suffixe
5	2	1	5

Figure 4 : Table de configuration de l'identifiant et du mot de passe

Une fois la configuration terminée par le gestionnaire, les utilisateurs peuvent être crées. Dans le formulaire de saisie des informations personnelles, il convient de sélectionner un type de génération d'identifiant et de mot de passe parmi les trois options proposées :

- Automatique
- Semi-automatique
- Manuel

Lorsque la méthode de gestion est automatique, l'utilisateur ne saisit aucun des deux champs du formulaire dédiés à l'identifiant et au mot de passe. Ils sont tous deux remplis automatiquement en tenant compte des paramètres de la table de configuration.

Lorsque la méthode de gestion est semi-automatique, l'utilisateur ne saisit que son mot de passe et la confirmation de celui-ci. Ce dernier doit impérativement commencer par

une majuscule et comporter au minimum 8 caractères. L'identifiant est généré automatiquement en tenant compte des paramètres de la table de configuration.

Enfin, lorsque la méthode de gestion est manuelle, l'utilisateur saisit l'identifiant et le mot de passe de son choix. Le mot de passe possède les mêmes contraintes qu'en mode semi-automatique. De plus, un contrôle est effectué à la validation de manière à vérifier qu'il n'y ait pas de doublon de l'identifiant dans l'annuaire. Si c'est le cas, une alerte bloquante est affichée à l'écran et l'utilisateur est invité à réitérer la saisie.

La partie suivante détaille au niveau technique le lien entre la configuration choisie et la génération des champs « Identifiant » et « Mot de passe » lors de la création d'un utilisateur.

3 Rapport technique

Cette partie détaille les choix de conception effectués, le déroulement de la réalisation du projet ainsi que les bonnes pratiques de programmation respectées. Pour rappel, l'objectif final est d'obtenir un outil de génération d'identifiant et de mot de passe paramétrable.

3.1 Conception fonctionnelle

Comme évoqué dans la partie précédente, la création d'un utilisateur nécessite deux étapes préalables : la configuration des paramètres liés à la génération de l'identifiant et du mot de passe par le gestionnaire puis la création du processus nommé « Gestion des utilisateurs ». Ces deux étapes sont réalisées depuis l'application créée dans le studio Moovapps nommée « Extranet Client ». Le fonctionnement global de l'application, illustré à la figure 5, peut être décrit de la manière simplifiée suivante : dans un premier temps, un client est créé dans l'annuaire grâce au processus « Gestion des clients » puis dans un second temps, des utilisateurs lui sont associés (comme le schématise la figure 1). Lors de cette phase consistant à associer un ou plusieurs utilisateurs, un second processus, nommé « Gestion des utilisateurs », intégré au premier, est appelé à chaque nouvel utilisateur créé.

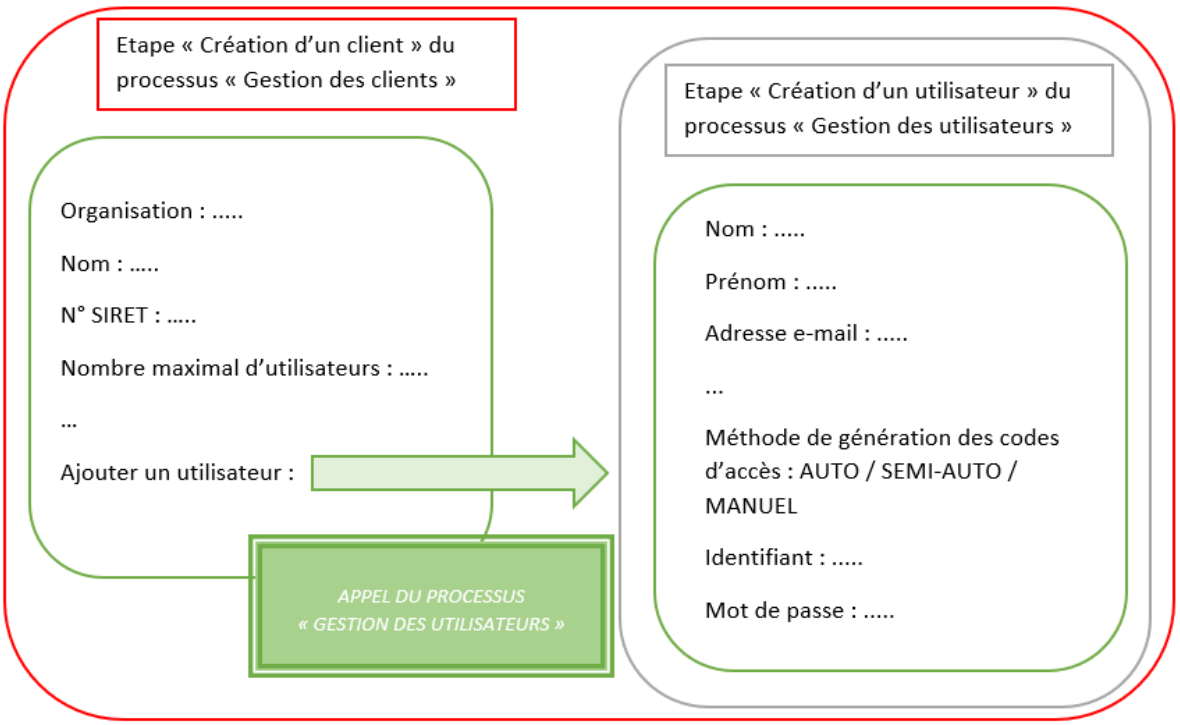

Figure 5 : Schéma simplifié du fonctionnement global de l'application « Extranet Client »

Le processus « Gestion des utilisateurs » possède deux grandes étapes : la création et le suivi de l'utilisateur. Le suivi permet de gérer les informations concernant cet utilisateur. C'est l'étape « Création » qui assure la génération d'un identifiant et d'un mot de passe paramétrables.

La création d'un compte utilisateur s'effectue via un formulaire positionné à cette étape du processus. Ce dernier comporte différentes informations personnelles à saisir manuellement dont le nom, le prénom et l'adresse e-mail qui sont susceptibles d'être utilisées pour la génération de l'identifiant. Ce formulaire contient également les champs « Identifiant » et « Mot de passe » qui peuvent être saisis manuellement ou automatiquement.

Les fonctionnalités standards du studio Moovapps ne permettent pas de gérer la complexité de la demande. Pour contourner ce problème, il est possible de positionner des classes d'extension Java sur les formulaires d'étapes, dans le diagramme de workflow du processus depuis le studio. Ces dernières contiennent du code permettant de gérer les comportements spécifiques non pris en charge par le standard. Comment ces classes d'extension Java assurent-elles la liaison entre la table de configuration éditée par le gestionnaire et le formulaire à la création d'un nouvel utilisateur pour remplir automatiquement les champs « Identifiant » et « Mot de passe » ? C'est l'objet de la partie suivante.

3.2 Conception technique

Cette partie technique détaille le développement du code java qui permet, au travers de différentes méthodes, la génération d'un identifiant ainsi que d'un mot de passe aléatoire en prenant compte des paramètres définis de manière fonctionnelle par le gestionnaire dans la table de configuration de l'application « Extranet Client ».

Celles-ci sont appelées lors d'évènements définis grâce à des méthodes java. La classe d'extension « GenerateLoginAndPassword » est appelée à chaque modification du champ du formulaire « Méthode de génération de l'identifiant et du mot de passe » grâce à la méthode onPropertyChanged (IProperty prop). La classe IProperty est, comme son nom le suggère, une interface. Elle permet de manipuler un objet correspondant à un champ de formulaire. Pour rappel, le champ qui va être évalué ici est de type « bouton radio » et comporte trois choix possibles : automatique, semi-automatique, ou manuel. Le détail du comportement de chacune de ces trois options a été rédigé dans la partie 2. C'est donc à l'intérieur de cette classe d'extension, mais également à l'intérieur de cette méthode que le code de génération d'identifiant et de mot de passe est positionné.

A noter que d'autres classes d'extensions (visibles sur la figure 6) sont présentes sur le diagramme de workflow. Leur rôle est de « contrôler » la cohérence des données saisies, notamment lors du paramétrage. Tout le contenu du code de ces classes est disponible en annexes techniques mais ne sera pas entièrement détaillé dans ce rapport. Cependant, la

partie 3.2.1 se terminera par un bref tour d'horizon survolant la fonction principale de chacune d'elles.

3.2.1 Technologies utilisées

Avant de s'intéresser aux détails techniques du code, il convient de nommer les différents outils utilisés lors de la réalisation de ce développement.

Lorsque l'on crée un client, un utilisateur, un document de processus ou n'importe quelle autre entité dont l'objectif est d'être stockée, il est nécessaire d'utiliser des bases de données que l'on rattache à Moovapps lors de l'installation initiale. Le SGBD (Système de Gestion de Bases de Données) utilisé lors du développement de l'application est MySQL et la manipulation des données stockées en bases est effectuée depuis le logiciel « Heidi SQL ».

Le code est rédigé en Java depuis l'IDE (en français : environnement de développement) « Eclipse ». Ce dernier permet de démarrer/arrêter différents serveurs personnalisés paramétrables pour lesquels on définit le chemin de stockage système du serveur d'application principal, ici « **JBoss** », un port web pour l'accessibilité de l'application en local, une JRE (en français : l'environnement d'exécution Java) ainsi qu'un ou plusieurs projets. Un serveur d'application est plus complet qu'un serveur web. Pour rappel, un serveur web permet d'interpréter les requêtes du client et de lui afficher en réponse soit des pages html statiques soit des pages dynamiques via du code PHP. Dans ce second cas, on dit que le serveur web est dynamique. Le serveur d'application, quant à lui, fournit des services aux applications, tels qu'un accès aux bases de données ou encore un service d'authentification. Il offre également un contexte d'exécution pour des composants applicatifs, ce qui facilite le déploiement du code.

L'architecture du code depuis Eclipse est illustrée à la figure 6.

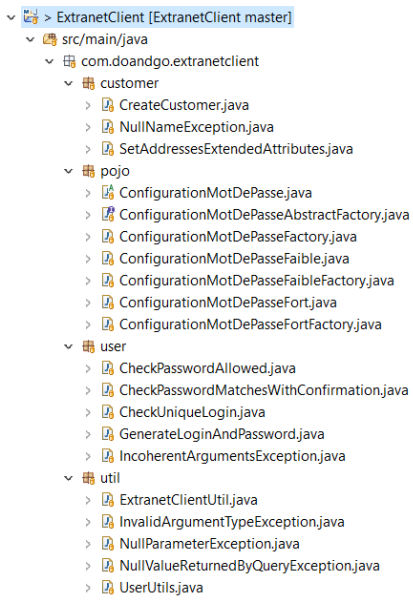

Figure 6 : Architecture des packages et des classes java depuis Eclipse

Le code est structuré en quatre packages. « customer » est dédié à la création du client, il n'est pas détaillé dans ce rapport. « user » est dédié à la création de l'utilisateur. « **pojo** » a été créé dans le but de manipuler un objet java de type personnalisé ConfigurationMotDePasse créé à l'aide d'une **fabrique abstraite**. En règle générale, l'utilisation de pojos facilite les tests et assure la propreté du système. Le package « util » comporte des constantes et des méthodes génériques qui peuvent être appelées dans plusieurs classes. A noter que chaque package gère ses propres exceptions.

Le package « user » contient trois classes dont le nommage commence par « Check ». Ces classes d'extension sont positionnées sur la table de configuration d'identifiant et de mot de passe dans le but de vérifier la cohérence des données saisies par le configurateur. Ces vérifications effectuées en amont de la génération d'identifiant permettent de détecter une éventuelle erreur au plus tôt et de lancer une alerte bloquante empêchant ainsi au configurateur de valider la configuration.

La classe « CheckPasswordAllowed » vérifie que le mot de passe saisi, dans le cas où le mode de saisie est manuel ou semi-automatique, est bien conforme aux règles de sécurité minimales définies dans le code. Le mot de passe doit commencer par une majuscule et comporte au minimum huit caractères.

La classe « CheckPasswordMatchesWithConfirmation » vérifie si la confirmation saisie, dans le cas où le mode de saisie est manuel ou semi-automatique, correspond bien au mot de passe saisi dans le champ au-dessus.

La classe « CheckUniqueLogin » vérifie si l'identifiant, dans le cas où il est saisi manuellement, n'est pas déjà existant dans l'annuaire.

Un possible scénario erroné est illustré par le diagramme de séquences de la figure 7.

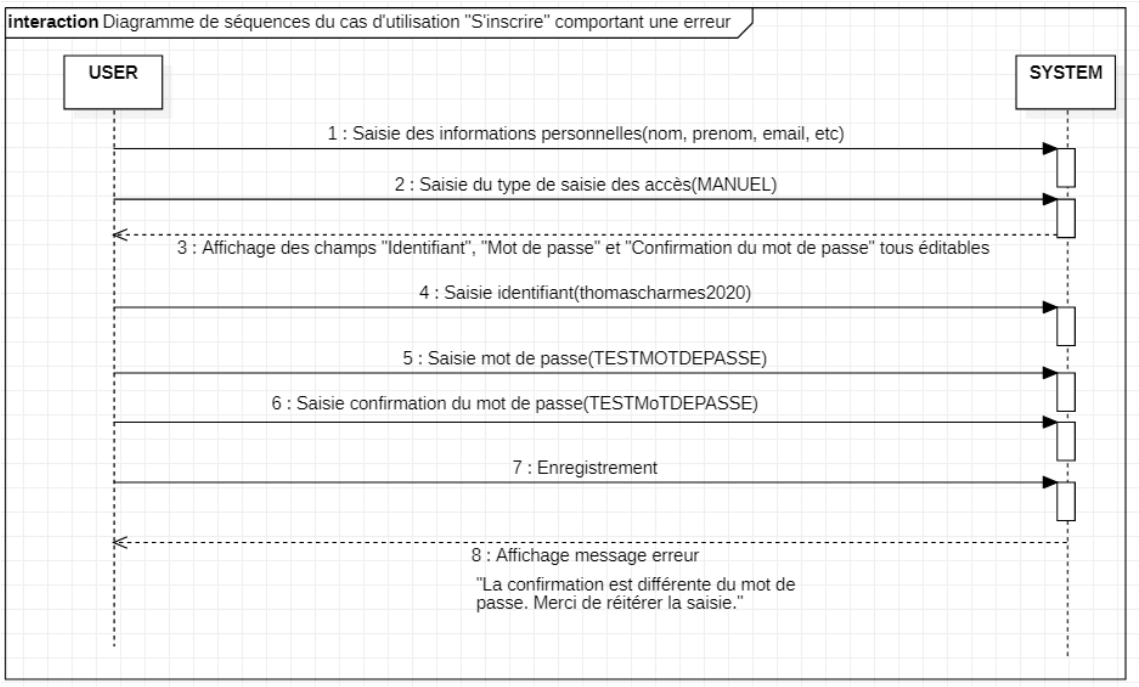

Figure 7 : Diagramme de séquences du cas d'utilisation "S'inscrire" comportant une erreur

A noter que la vérification sur la non-nullité des champs n'apparaît pas dans le code car ce comportement est géré par le standard Moovapps. En effet, ce dernier permet de rendre des champs obligatoires. Ces trois classes sont destinées à empêcher une utilisation inappropriée de l'application. Le code de celles-ci n'est pas détaillé dans ce rapport qui ne se focalise pas sur l'aspect manuel de la création de l'utilisateur mais plutôt sur l'aspect automatique. Cependant, le code complet des ces trois classes est disponible en annexes techniques.

3.2.2 Gestion de l'identifiant

La gestion automatique de l'identifiant s'effectue depuis la classe « GenerateLoginAndPassword », contenue dans le package « user » et dont le code complet se situe en annexe A. Cette partie détaille les méthodes destinées à générer un identifiant.

Comme évoqué dans les parties précédentes, il est possible de sélectionner un modèle d'identifiant parmi trois possibilités, la première est de se baser sur le prénom et le nom de l'utilisateur. Lorsque ce dernier sélectionne le mode de gestion « automatique » ou « semi-automatique », le champ « Identifiant » est rempli automatiquement par le système. Cela est possible grâce aux appels successifs des méthodes présentes dans le diagramme de séquences de la figure 8.

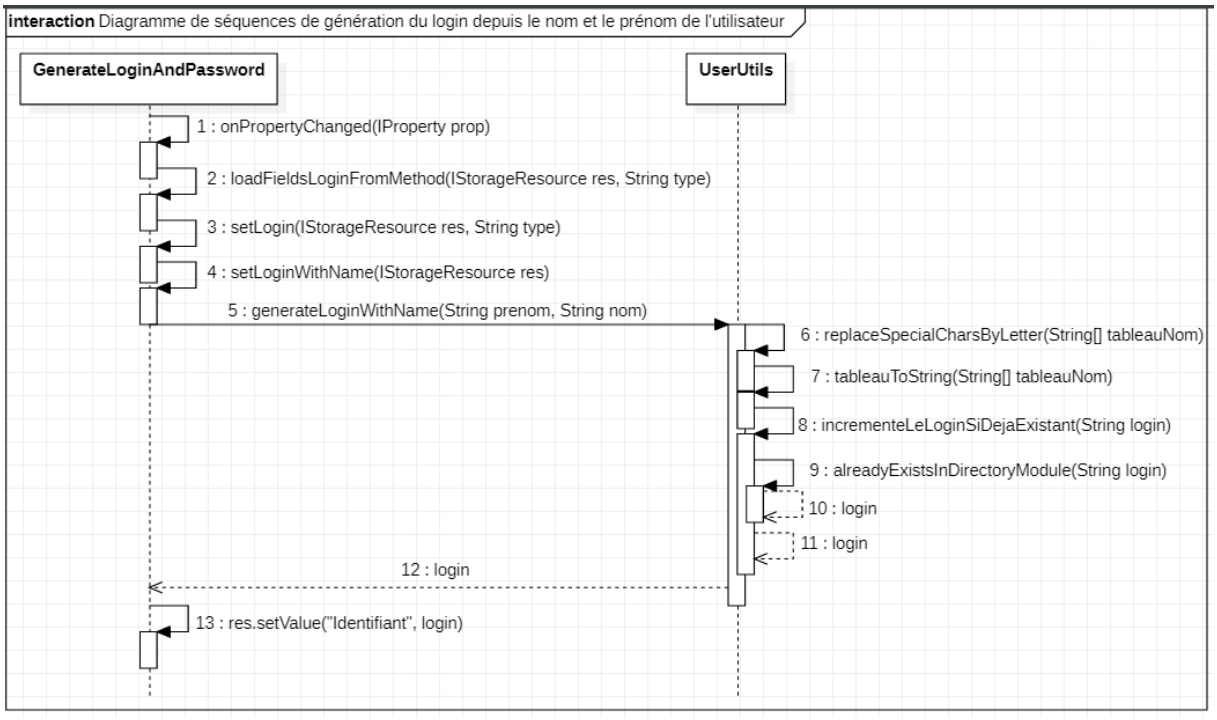

Figure 8 : Diagramme de séquences de génération de l'identifiant à partir du nom et prénom de l'utilisateur

Les méthodes 1, 2 et 3 sont communes aux trois types possibles de génération d'identifiant. La méthode principale de ce diagramme, generateLoginWithName (String prenom, String nom), est la principale. Son code est illustré à la figure 9.

```java
/**
 * crée un login de la forme initiale du prénom + nom sans les caractères
 * spéciaux
 *
 * @param prenom
 * @param nom
 * @return
 * @throws NullArgumentException
 * @throws NullParameterException
 * @throws InvalidArgumentTypeException
 */
public static String generateLoginWithName(String prenom, String nom)
        throws NullArgumentException, NullParameterException, InvalidArgumentTypeException {

    if (NullUtils.isAtLeastOneNullOrEmpty(prenom, nom))
        throw new NullParameterException("Le nom ou le prénom est une chaîne de caractères nulle ou vide");

    prenom = prenom.trim().toLowerCase();
    nom = nom.trim().toLowerCase();

    if (!RegexpUtils.isStartByLowercase(prenom) || RegexpUtils.hasNumber(nom))
        throw new InvalidArgumentTypeException(
                "Le nom ou l'initiale du prénom comporte au moins un chiffre");

    String login = null;
    Character initialeDuPrenom = prenom.charAt(0);
    String[] tableauNom = nom.split("");
    tableauNom = replaceSpecialCharsByLetter(tableauNom);
    String nomSansAccent = tableauToString(tableauNom);
    login = initialeDuPrenom + nomSansAccent;
    try {
        login = incrementeLeLoginSiDejaExistant(login);
    } catch (DataBaseConnectionException | NullValueReturnedByQueryException e) {
        reportError("Erreur lors de l'incrémentation du login : " + e.getMessage());
        e.printStackTrace();
    }
    return login;
}
```

Figure 9 : Code source de la méthode generateLoginWithName(String prenom, String nom)

Cette méthode peut être décomposée en trois étapes. La première est la vérification de la cohérence des arguments et leur formatage. A l'origine, le prénom et le nom doivent être tous les deux non-nuls et ne comporter que des lettres avec ou sans accent. Le trait d'union est le seul caractère spécial autorisé. Le formatage consiste à supprimer les espaces grâce à la méthode trim() et à convertir les deux paramètres en minuscules. La deuxième étape est dédiée au traitement des données. Tout d'abord, son rôle est d'extraire l'initiale du prénom grâce à la méthode charAt(0) appliquée à un String qui retourne un objet de type Character. Il faut également remplacer les éventuels accents présents dans le nom par la lettre correspondante sans accent. Cela se fait en utilisant la méthode replaceSpecialCharsByLetter(String[] tableauNom) qui remplace chaque élément du tableau s'il comporte un accent et qui supprime l'élément si ce dernier est un trait-d'union. La méthode tableauToString(String[] tableauNom) converti le tableau en une chaîne de caractère qui représente le nom sans accent. La création d'une variable de type chaîne de caractère nommée « login » permet de concaténer l'initiale du prénom avec le nom sans accent. Enfin, la troisième étape est de vérifier que cet identifiant n'est pas déjà existant dans l'annuaire. Pour cela, la méthode incrementeLeLoginSiDejaExistant(String login) est appelée. Son code source est illustré par la figure 10. A noter que cette méthode utilise les expressions régulières appelées **Regex**.

```
private static String incrementeLeLoginSiDejaExistant(String loginInitial)
        throws DataBaseConnectionException, NullValueReturnedByQueryException {
    int i = 1;
    String login = loginInitial;
    while (alreadyExistsInDirectoryModule(login)) {
        login = loginInitial + Integer.toString(i);
        i++;
    }
    return login;
}

public static boolean alreadyExistsInDirectoryModule(String login) {
    IDirectoryModule dm = null;
    try {
        dm = Modules.getDirectoryModule();
        IUser userByLogin = dm.getUserByLogin(login);
        if (! NullUtils.isNull(userByLogin))
            return true;
    }
    finally {
        Modules.releaseModule(dm);
    }
    return false;
}
```

Figure 10 : Code source des méthodes incrementeLeLoginSiDejaExistant(String login) et alreadyExistsInDirectoryModule(String login)

Le fonctionnement de cette méthode est simple, ajouter une valeur numérique croissante aux identifiants déjà existants. Par exemple, s'il y a déjà un utilisateur dans l'annuaire dont l'identifiant est « tcharmes », le prochain utilisateur qui s'appelle Théo Charmes qui veut créer un compte aura pour identifiant « tcharmes1 ». Puis Tatiana Charmes aura pour identifiant « tcharmes2 » et ainsi de suite. Comment vérifier qu'un identifiant est déjà existant dans l'annuaire ? Grâce à la méthode alreadyExistsInDirectoryModule(String login). Le directory module est l'annuaire de Moovapps. La javadoc Moovapps (voir bibliographie) propose la méthode getUserByLogin(String login) qui renvoie un utilisateur de type IUser. Si cette méthode renvoie une valeur nulle, alors l'utilisateur n'existe pas. En d'autres termes, l'identifiant est disponible.

A noter qu'il est également possible de vérifier l'existence d'un utilisateur directement dans la base de données, en passant par la connexion externe. Cela est possible grâce à la méthode alreadyExistsInDataBase(String login) qui se trouve dans la classe « UserUtils » disponible en annexe B. Cela explique les exceptions prises en charge par la méthode incrementeLeLoginSiDejaExistant(String login).

La seconde option de génération d'identifiant est de se baser sur la société de l'utilisateur. Dans ce cas-là, la méthode appelée est generateLoginWithSociete(String prefixeSociete, Float indexDepart, Float nbChiffresSuffixe) dont le code est illustré par la figure 11.

```java
public static String generateLoginWithSociete(String prefixeSociete, Float indexDepart, Float nbChiffresSuffixe)
        throws NullArgumentException, InvalidArgumentTypeException, DataBaseConnectionException,
        NullValueReturnedByQueryException, InvalidArgumentValueException {

    if (NullUtils.isNullOrEmpty(prefixeSociete))
        throw new InvalidArgumentTypeException("La société est vide");

    if (! indexContientMoinsDeChiffresQueLeSuffixe(indexDepart.intValue(), nbChiffresSuffixe.intValue())) {
        throw new InvalidArgumentValueException("L'index de départ contient plus de chiffres que le suffixe souhaité");
    }

    int i = indexDepart.intValue();
    String login = prefixeSociete + "-";
    String suffixe = StringUtils.cutAndFill(Integer.toString(i), nbChiffresSuffixe.intValue(), true, "0", false);

    while (alreadyExistsInDatabase(login + suffixe)) {
        i++;
        suffixe = StringUtils.cutAndFill(Integer.toString(i), nbChiffresSuffixe.intValue(), true, "0", false);
    }

    if (! indexContientMoinsDeChiffresQueLeSuffixe(i, nbChiffresSuffixe.intValue())) {
        throw new InvalidArgumentValueException("L'index contient plus de chiffres que le suffixe souhaité");
    }

    return login + suffixe;
}
```

Figure 11 : Code source de la méthode generateLoginWithSociete(String prefixeSociete, Float indexDepart, Float nbChiffresSuffixe)

L'objectif principal de cette méthode est de générer un identifiant qui prend en compte le nom de la société, par exemple « DO&GO », et y ajoute un trait d'union suivi d'une valeur numérique incrémentale qui varie selon l'existence de l'identifiant dans la base de données. La différence avec l'incrément de la méthode generateLoginWithName(String prenom, String nom) est que dans ce cas, il est possible de déterminer une valeur de départ de l'incrément qui ne sera donc plus « 1 » par défaut. Il est également possible de déterminer le nombre exact de « 0 » à afficher avant la valeur numérique de l'incrément. Cela est possible en utilisant la méthode cutAndFill(String mot, int nombreDeCaracteres, boolean remplir, String caractereAInserer, boolean apres) qui modifie la chaîne de caractères « mot » et y insérant le caractère « caractereAInserer » avant ou après, autant de fois que « nombreDeCaracteres ». Par exemple : cutAndFill(1, 3, true, « 0 », false) retourne la chaîne de caractères « 0001 ». La valeur finale de l'identifiant est alors « DO&GO-0001 ».

A noter qu'ici les paramètres numériques sont de type Float alors que ce sont des nombres entiers, car Float est le type par défaut d'un champ numérique dans un formulaire Moovapps.

La troisième est dernière option de génération d'identifiant consiste à utiliser l'adresse e-mail de l'utilisateur. Dans ce cas précis, aucun traitement spécifique n'est effectué sur l'adresse e-mail. Une adresse e-mail ne peut donc être associée qu'à un seul et unique compte.

La classe « GenerateLoginAndPassword » gère donc la génération d'identifiant personnalisé et paramétrable mais également la génération d'un mot de passe. Cela est possible grâce à des méthodes dont le comportement est détaillé dans la partie suivante.

3.2.3 Gestion du mot de passe

La génération du mot de passe s'effectue en plusieurs étapes. Dans un premier temps, un objet de type ConfigurationMotDePasse est créé grâce à l'appel de la méthode getConfigurationMotDePasse(Factory). Cet objet est appelé **« bean »**. Un bean peut être défini par une classe publique, comportant un constructeur par défaut public et sans paramètres, ou défini automatiquement s'il ne l'est pas par l'utilisateur. Il ne contient pas de champs publics mais des propriétés privées accessibles depuis des getters et setters. Ce bean est créé depuis une fabrique abstraite, contient des propriétés qui sont des paramètres récupérés depuis la table de configuration. Ils représentent le nombre de caractères spéciaux, le nombre de chiffres, le nombre de majuscules, le nombre de minuscules, ou le nombre total de caractères contenus dans le mot de passe souhaité. L'objet qui contient ces éléments est passé en paramètre de la méthode generateRandomStrongPassword(ConfigurationMotDePasse config) qui est la méthode principale. L'utilisation d'une fabrique abstraite permet de ne pas préciser le type de configuration à passer en paramètre. Ce type est soit ConfigurationMotDePasseFaible (ce qui correspond dans la configuration au choix : mot de passe simple) soit ConfigurationMotDePasseFort (correspondant à un mot de passe sécurisé). Le fonctionnement global de la génération d'un mot de passe sécurisé est illustré à la figure 12.

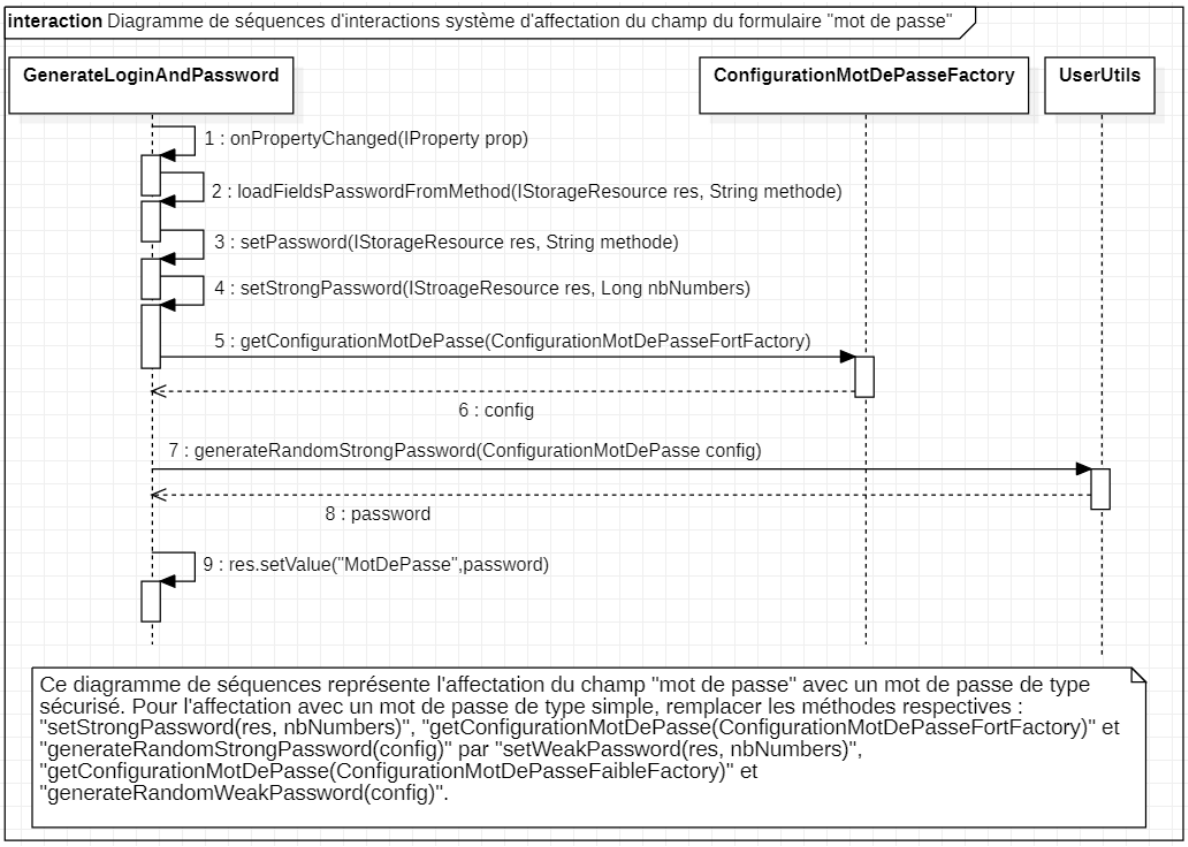

Figure 12 : Diagramme de séquences d'affectation du champ "Mot de passe" du formulaire

La méthode generateRandomStrongPassword(ConfigurationMotDePasse config) a évolué. En effet, la frise chronologique de la figure 13 représente ses quatre versions différentes :

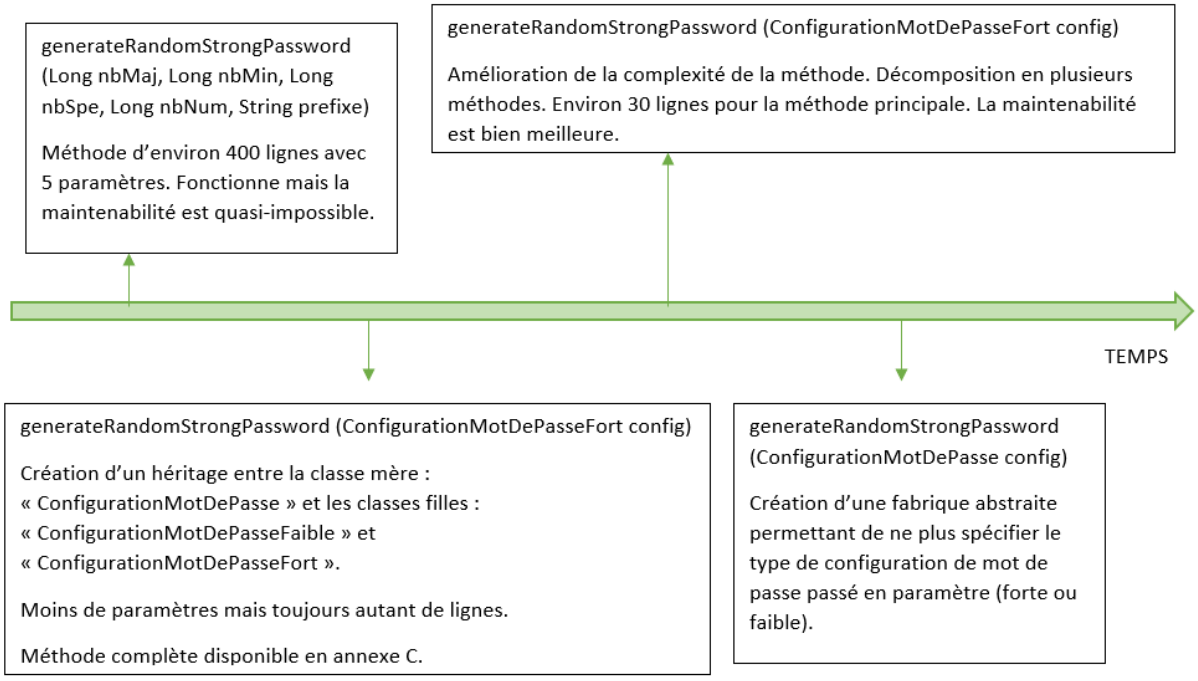

Figure 13 : Frise chronologique représentant l'évolution de la méthode generateRandomStrongPassword(xxx) au fil du temps

La première version comportait de nombreux paramètres, ce qui n'est pas une bonne pratique de programmation si l'on en croit l'ouvrage « Clean code » de R. Martin. C'est pourquoi, pour remédier à ce problème, la deuxième version comportait un héritage simple, illustré par le diagramme de classes de la figure 14. Cette deuxième méthode, dont le code complet se trouve en annexe C, possède deux principaux inconvénients. D'une part, le type du paramètre varie selon le type de mot de passe souhaité, ce qui restreint la généricité de la méthode. D'autre part, la méthode est longue, très longue, environ quatre cent lignes de « if » imbriqués et est donc quasi-impossible à maintenir.

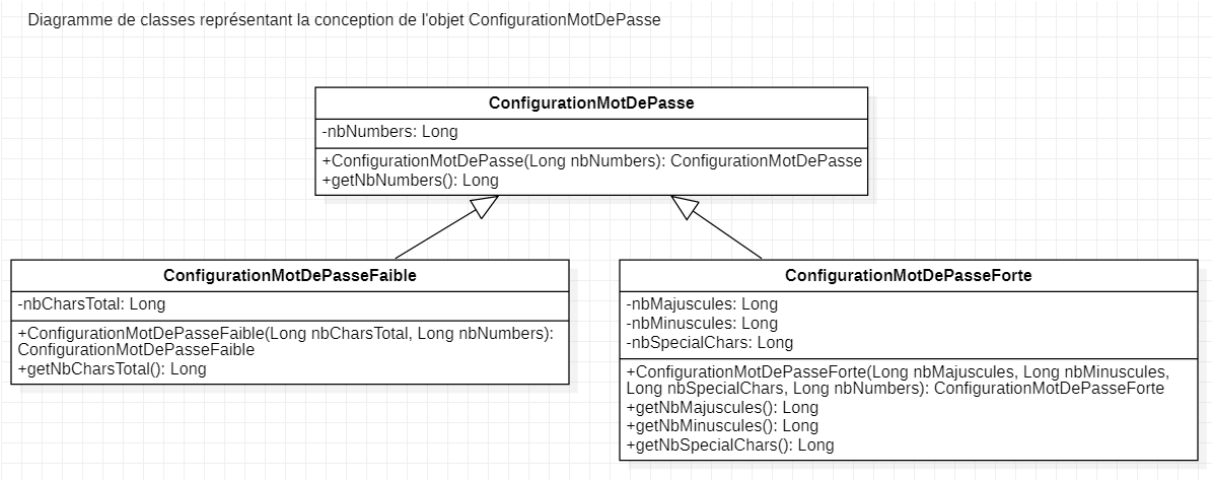

Figure 14 : Diagramme de classes représentant l'héritage utilisé pour définir un type de configuration de mot de passe

Pour remédier à ce problème de longueur et de maintenabilité, la troisième version de la méthode a été créée. Son fonctionnement est décrit par le diagramme de séquences système de la figure 15.

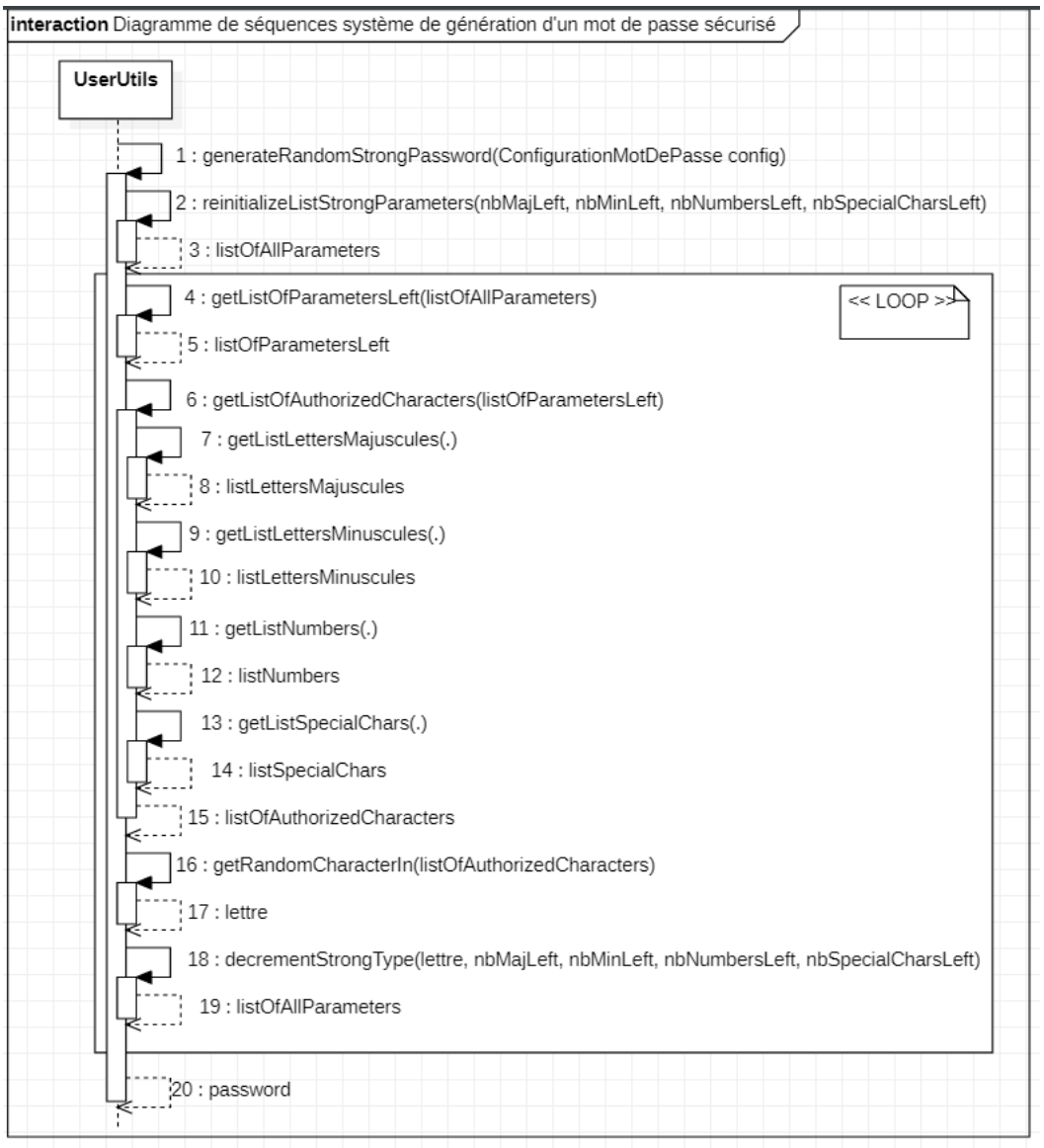

Figure 15 : Diagramme de séquences de génération d'un mot de passe sécurisé

L'idée principale ici est de faire un tirage au sort d'un caractère à la fois uniquement parmi les caractères encore autorisés. Pour cela, la première chose est d'initialiser (puis de réinitialiser avant chaque nouveau tirage d'un caractère) la liste des paramètres. La méthode getListOfParametersLeft(String[] listOfAllParameters) permet de supprimer de la liste tous les paramètres qui ne peuvent plus être tirés au sort, c'est-à-dire ceux pour lesquels le nombre de caractères restants à tirer est égal à zéro. Depuis cette liste de paramètres restants, il est possible de créer une liste qui contient tous les caractères qu'il est possible de tirer au sort. Il suffit de tirer un caractère au sort dans cette liste grâce à la méthode getRandomCharacterIn(String[] listOfAuthorizedCharacters), puis en fonction du

type de caractère qui a été tiré, décrémenter le nombre de tirage restant pour ce type, et enfin réitérer cette opération autant de fois qu'il n'y a de lettres restantes.
Voici un exemple pour illustrer le fonctionnement de cette méthode :

Si le mot de passe doit avoir seulement 1 majuscule et 1 chiffre (ce n'est pas ce que l'on peut appeler un mot de passe sécurisé mais c'est pour illustrer plus simplement le comportement de la méthode), au début les données sont les suivantes :
nbMajLeft = 1, nbMinLeft = 0, nbSpecialCharsLeft = 0, nbNumbersLeft = 1.
La liste des paramètres restants est donc égale à [nbMajLeft, nbNumbersLeft]. Ce qui correspond à la configuration initiale.
Une liste des tirages possibles pour ce prochain caractère spécifique est alors égale à [abcdefghijklmnopqrstuvwxyz0123456789]. La méthode getRandomCharacterIn(list) renvoie par exemple « t ». La valeur nbMinLeft est alors décrémentée et est désormais égale à 0. Lors de la réinitialisation des paramètres restants, la liste renvoyée sera donc égale à [nbNumbersLeft]. Le second caractère sera donc tiré parmi la liste [0123456789]. Cela permet de tirer au sort systématiquement un caractère du ou de l'un des types attendus. Par exemple, le deuxième caractère peut être « 2 ». Le mot de passe final sera alors « t2 » et il respecte bien la configuration initiale qui était 1 minuscule et 1 chiffre.

Reste à régler le problème de typage du paramètre de la méthode. Pour cela, la quatrième et dernière version de la méthode a été créée. Celle-ci contient une fabrique abstraite dont le diagramme de classes se situe en figure 16.

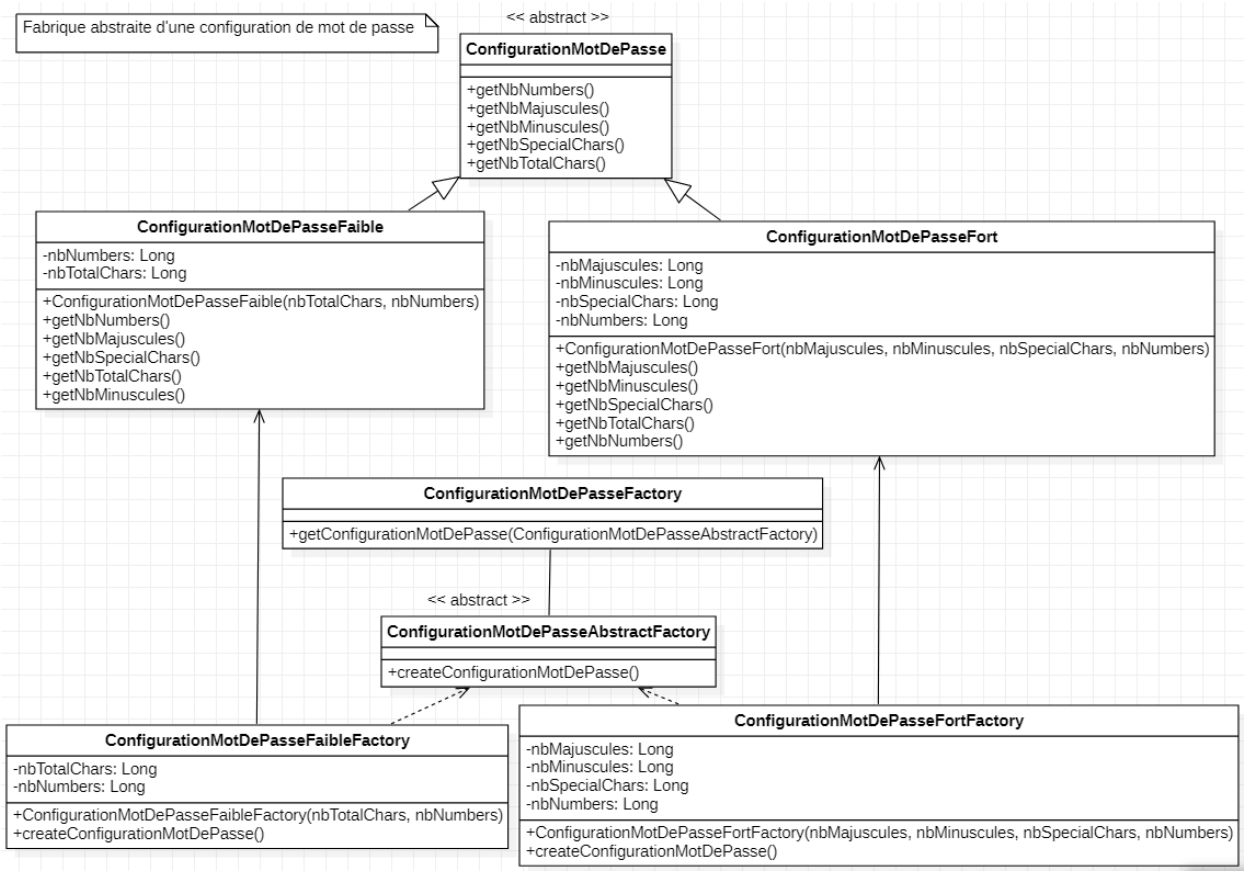

Figure 16 : Diagramme de classes de la fabrique abstraite utilisée pour créer une configuration de mot de passe

Il était important de faire ces modifications de manière à optimiser les performances de l'application en réduisant le temps d'exécution de la méthode. Ces améliorations successives doivent être faites dès que possible, car en programmation si l'on en croit la loi de Leblanc : « Plus tard signifie jamais ».

Voici donc en figure 17 le code définitif de la méthode, dont le comportement est défini dans l'exemple ci-dessus :

```
public static String generateRandomStrongPassword(ConfigurationMotDePasse config) {
    // Récupération des éléments de la configuration :
    Long nbMajLeft = config.getNbMajuscules();
    Long nbMinLeft = config.getNbMinuscules();
    Long nbNumbersLeft = config.getNbNumbers();
    Long nbSpecialCharsLeft = config.getNbSpecialChars();
    Long nbCharactersLeft = config.getNbTotalChars();
    String password = "";
    String lettre = "";
    Map<String, Long> listOfAllParameters = reinitializeListStrongParameters(nbMajLeft, nbMinLeft, nbNumbersLeft,
            nbSpecialCharsLeft);

    while (nbCharactersLeft > 0) {
        Map<String, Long> listOfParametersLeft = getParametersLeft(listOfAllParameters);
        List<String> listOfAuthorizedCharacters = getListOfAuthorizedCharacters(listOfParametersLeft);
        lettre = getRandomCharacterIn(listOfAuthorizedCharacters);
        password += lettre;
        listOfAllParameters = decrementStrongType(lettre, nbMajLeft, nbMinLeft, nbNumbersLeft, nbSpecialCharsLeft);
        for (Entry<String, Long> param : listOfAllParameters.entrySet()) {
            if (param.getKey().equals("nbMajLeft"))
                nbMajLeft = param.getValue();
            if (param.getKey().equals("nbMinLeft"))
                nbMinLeft = param.getValue();
            if (param.getKey().equals("nbNumbersLeft"))
                nbNumbersLeft = param.getValue();
            if (param.getKey().equals("nbSpecialCharsLeft"))
                nbSpecialCharsLeft = param.getValue();
        }
        nbCharactersLeft--;
    }
    return password;
}
```

Figure 17 : code définitif de la méthode generateRandomStrongPassword(ConfigurationMotDePasse config)

Les méthodes inclues ici respectent d'avantage le **principe de responsabilité unique** (Single Responsability Principle en anglais) selon lequel "Une fonction ne doit faire qu'une seule chose. Elle doit la faire bien et ne faire qu'elle". (Martin, 2009, p.40)

Elles respectent également le principe dit de **moindre surprise** selon lequel toute méthode ou classe doit implémenter les comportements auxquels un autre programmeur peut raisonnablement s'attendre.

Le code source des méthodes appelées est disponible en annexe B. Le comportement de la méthode generateRandomWeakPassword(ConfigurationMotDePasse config), est fortement similaire. La différence majeure est le nombre de paramètres qui est moins élevé. De plus, chaque mot de passe dit « faible » commence par une Majuscule. Le code source de cette méthode est également disponible en annexe B.

3.2.4 Principes de programmation respectés

Le livre « Clean code » de R. Martin recense une multitude de bonnes pratiques de programmation à respecter pour obtenir un code le plus propre possible. Certains de ces principes, répertoriés ci-dessous, ont été respectés.

Tout d'abord, la gestion de l'internationalisation. Certains clients ont un moovapps configuré en anglais. Le fichier de configuration doandgo-extranetclient.xml dont le code se trouve en annexe J contient les clés de traductions des messages d'alerte envoyés à l'écran en cas d'erreur. C'est la locale qui définit par défaut le langage utilisé.

De multiples opérations de **refactoring** ont été effectuées afin de respecter la **règle de décroissance**, stipulant que le code doit pouvoir être lu de haut en bas comme un livre. Cela concorde avec le principe **d'affinité conceptuelle** signifiant que les concepts étroitement liés au niveau conceptuel doivent se trouver proches physiquement les uns des autres. Par exemple dans la classe « UserUtils », on remarque que le code est séparé en deux blocs implicites, l'un consacré à la génération de l'identifiant et l'autre à la génération du mot de passe. A noter que le code testé après chaque opération de refactoring afin de garantir son bon fonctionnement malgré les modifications. Pour rappel, le ré-usinage de code ne doit en aucun cas impacter le déroulement du programme.

Le code de la classe UserUtils respecte le **principe ouvert/fermé** (Open – Close Principle en anglais) selon lequel le code doit être fermé à la modification mais ouvert aux évolutions. Si le gérant décide ultérieurement d'ajouter un quatrième type d'identifiant, il ne sera pas nécessaire de modifier les trois types déjà existants. Il suffira de créer une quatrième méthode adéquate.

Le nommage des méthodes a également été souvent repensé pour respecter les standards d'écriture (le **camelcase**) mais aussi pour respecter le **principe de Ward** que l'on peut résumer par cette phrase du livre « Clean code » de R. Martin : "Vous savez que vous travaillez avec du code propre lorsque chaque fonction que vous lisez correspond presque parfaitement à ce que vous attendiez."

D'autres **principes SOLID** ont également été respectés, comme le maintien de la cohésion notamment grâce à la création de variables d'instances placées au début de la classe GenerateLoginAndPassword, le **principe de substitution de Liskov** et le **principe de ségrégation d'interfaces** pour la configuration du mot de passe ou encore **le principe d'inversion des dépendances** qui stipule que les classes doivent dépendre d'abstractions, non de détails concrets. Cela est illustré avec la fabrique abstraite de configuration du mot de passe.

Les packages, classes, méthodes et variables comportent des noms significatifs qui sont non préfixées, lisibles, et prononçables. Ce qui évite la désinformation. A l'origine, les méthodes qui s'appelaient generateXXX étaient nommées createXXX mais ce mot est réservé pour désigner une opération en base de données d'après les **méthodes CRUD**.

Ainsi le code contient bien un mot par concept. On appelle le code "langage" pour qu'il soit un moyen de communication qui doit être compris facilement et structuré. Pour cela, les méthodes ont été autant que possible courtes, indentées, et faisant un minimum de traitement pour éviter les effets secondaires. Les méthodes ne contiennent pas trop d'arguments, la plupart sont niladiques, unitaires ou diadiques.

La gestion des exceptions est également un élément très important du code qui n'a pas été négligé, tout comme la rédaction de commentaires. Ils sont utiles mais non-redondants.

Au niveau visuel, l'espacement vertical entre les méthodes est uniforme et on observe une concentration verticale à l'intérieur de ces dernières pour une meilleure lisibilité. Horizontalement, une ligne ne dépasse pas la limite conseillée par les développeurs du célèbre « gang of four » de 120 caractères.

L'utilisation de variables de stockage a été utilisée. Cela permet d'éviter les **catastrophes ferroviaires**. Une catastrophe ferroviaire est une erreur causée par le nombre de méthodes les unes à la suite des autres trop important. Par exemple, il n'est pas recommandé d'écrire getX().getY().getZ() mais plutôt de décomposer l'écriture en x = getX() ; y = x.getY() ; z = y.getZ().

Enfin, le code respecte la **loi de Demeter** selon laquelle un module ne doit pas connaitre les détails d'un objet interne qu'il manipule.

Les **principes STUPID** (Singleton, Couplage fort, Non testable, Nommage indéchiffrable, Duplication) ont été évités autant que possible tout au long de la rédaction de ce code.

3.3 Utilisation finale du produit

Le résultat obtenu est conforme aux attentes : une application paramétrable qui permet la création d'un utilisateur dans l'annuaire Moovapps. Le gestionnaire définit des paramètres globaux communs à tous les futurs utilisateurs tels que le type d'identifiant sur lequel se baser ou le niveau de sécurité du mot de passe. Une fois cette configuration effectuée, la création d'un nouvel utilisateur s'effectue depuis un formulaire. Les informations personnelles requises sont le nom, le prénom, l'adresse, l'email, le numéro de téléphone, l'identifiant et le mot de passe. L'utilisateur a le choix de saisir manuellement ces deux derniers champs ou alors de demander à l'application de les saisir automatiquement.

Le déroulement d'une inscription est illustré par le diagramme de séquences de la figure 17. Deux visuels du formulaire sont illustrés aux figures 18, 19 et 20.

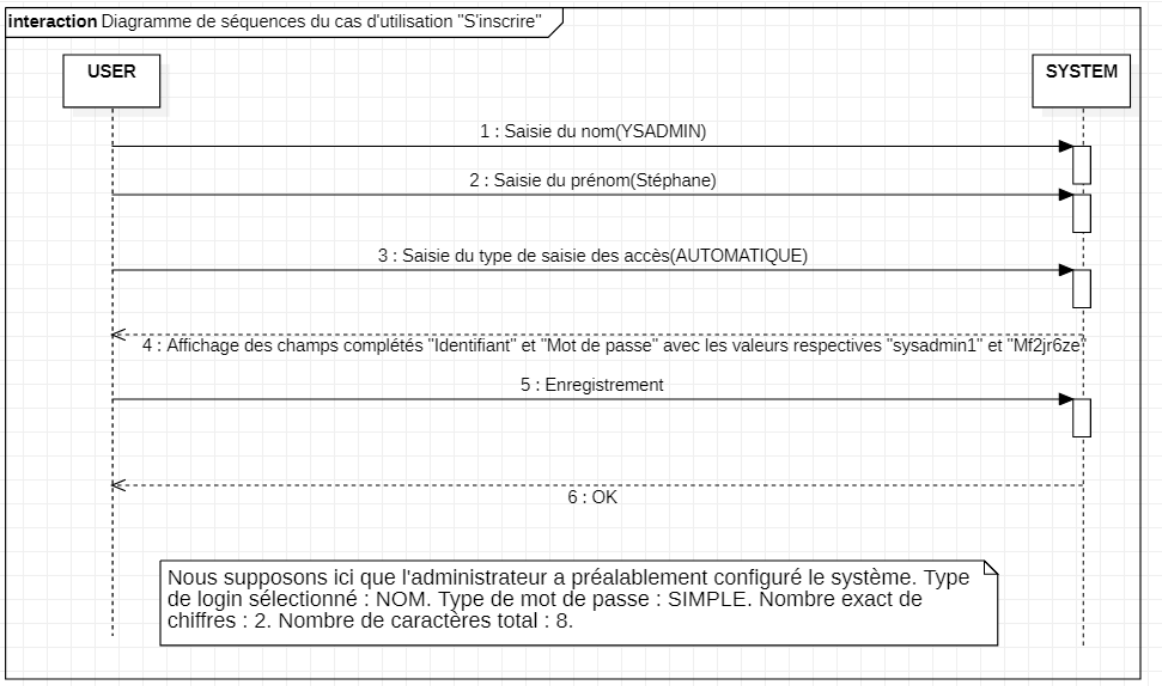

Figure 18 : Diagramme de séquences du cas d'utilisation "S'inscrire"

A noter que l'identifiant proposé par le système est « sysadmin1 » et non « sysadmin » étant donné qu'il ne peut y avoir deux identifiants identiques dans l'annuaire et que l'identifiant « sysadmin » existe par défaut pour l'administrateur système.

Figure 19 : Fragment du formulaire de création d'un utilisateur représentant la méthode de gestion automatique

Le résultat de la figure 18 montre que les champs « Identifiant » et « Mot de passe » ne sont pas éditables lorsque la méthode de gestion est automatique. Il suggère également que le gestionnaire a préalablement configuré l'application comme suit :

Type d'identifiant : Société

Nombre exact de chiffres du suffixe : 5

Index de départ : 1 et il existe déjà deux utilisateurs ? ou 3 ?

Type de mot de passe : Sécurisé

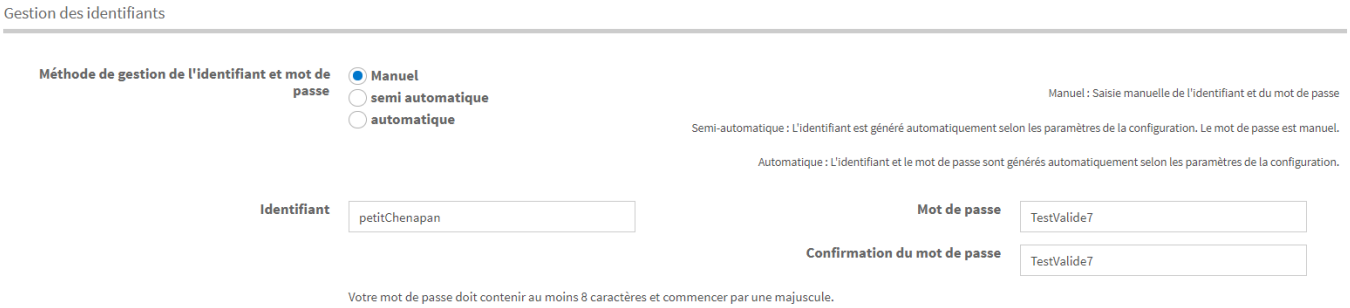

Figure 20 : Fragment du formulaire de création d'un utilisateur représentant la méthode de gestion manuelle

A noter que lorsque la méthode de gestion est manuelle, les champs « Identifiant », « Mot de passe » et « Confirmation du mot de passe » sont tous éditables. Lorsqu'elle est semi-automatique, les champs « Mot de passe » et « Confirmation du mot de passe » sont les seuls éditables.

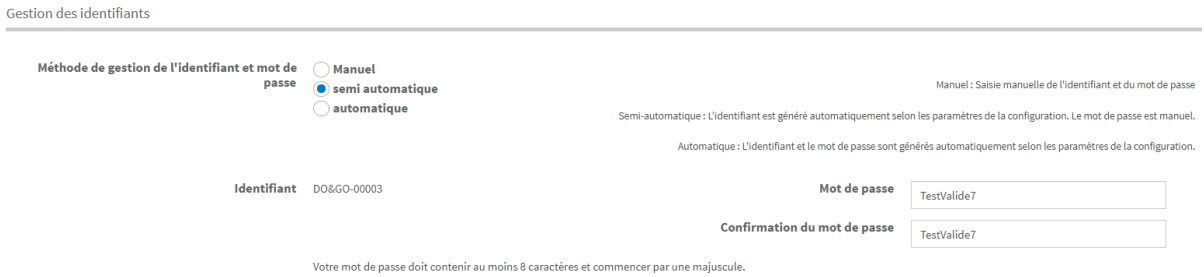

Figure 21 : Fragment du formulaire de création d'un utilisateur représentant la méthode de gestion semi-automatique

4 Gestion de projet

Cette partie détaille l'organisation globale de l'entreprise, son mode de fonctionnement ainsi que mon adaptation à celle-ci au niveau technique et social. Une sous-partie est consacrée à l'adaptation des conditions de travail en raison de la crise sanitaire du covid-19.

4.1 Démarche personnelle

4.1.1 Adaptation à l'entreprise

Lors de mon arrivée chez "Do&Go" à la fin du mois d'octobre, je me suis rapidement acclimaté à l'entreprise, notamment parce que j'ai eu à ma disposition un matériel adéquat. Cela s'est également traduit par la dispense d'une formation personnalisée aux outils informatiques utilisés au sein de l'entreprise par l'un des membres. Quant à l'adaptation sociale à l'entreprise, les pauses déjeuners en groupe et les activités externes ont également permis de favoriser mon intégration. J'ai reçu un encadrement adapté qui m'a rapidement fait me sentir bien, ce qui est primordial pour travailler dans un environnement commun.

Mon tuteur d'entreprise ayant également suivi une licence professionnelle d'informatique à l'IUT de Montpellier, il lui a été plus facilement possible de me donner des tâches à réaliser adaptées à l'avancement de la formation. De plus, il a suivi de près le déroulement de cette dernière en me demandant régulièrement mon ressenti. Cette bonne relation est notamment renforcée par une écoute constante de mes requêtes, qu'elles soient techniques ou matérielles

Un grand degré d'autonomie m'a été octroyé dès mon arrivée. L'idée globale que mon tuteur m'enseigne est que le cerveau retient mieux les informations qu'il a cherchées par lui-même. Le but est de monter progressivement en compétences. J'ai pour cela assisté à des réunions sans pour autant y participer. Cependant, je prends des initiatives en suggérant une solution particulière de développement à un problème donné. Cette autonomie se traduit également par le choix de mes horaires de travail.

Plusieurs démarches ont été utilisées pour trouver des informations. Tout d'abord, l'entreprise possède une base de ressources répertoriées dans un OneNote qui est documentée régulièrement, notamment avec les problèmes techniques rencontrés lors de développements pour pouvoir retrouver facilement la solution à ces mêmes problèmes en cas de nouvelles confrontations. L'outil Moovapps possède également une plateforme en ligne de "tickets" (un forum) accessible uniquement par des développeurs qui peuvent répondre à des questions. Moovapps possède également sa propre documentation. Enfin, pour la gestion de problèmes de développement, l'information manquante se trouve la plupart du temps dans la documentation Java.

4.1.2 Gestion de la crise sanitaire liée au Covid-19

Depuis la mi-mars 2020, la France a dû s'adapter aux mesures de restrictions en vigueur face à la pandémie du covid-19. En effet, un confinement total de toute la population a été mis en place pendant plus de deux mois, ce qui a contraint les entreprises à innover dans leur mode de fonctionnement. L'entreprise Do&Go n'a pas dérogé à cette règle.

Comme décrit dans la partie 4.1.1, il était déjà dans les habitudes de l'entreprise de pratiquer le télétravail. Un télétravail général a été décidé pour les quatre membres de l'équipe. Le contact était régulier, grâce à Google Hangout, nous organisions une réunion hebdomadaire le lundi matin, et je restais plus proche encore de monsieur Pontfort avec qui nous faisions des récapitulatifs tous les soirs. Ma montée en compétences aurait pu être ralentie par le manque de présence autour de moi mais grâce à des logiciels de prise en main du poste informatique à distance tels que « TeamViewer » ou « AnyDesk », ce n'a pas été le cas. Le télétravail dans des conditions exceptionnelles de confinement comporte des risques psychologiques, dont l'entreprise Do&Go est consciente. C'est pourquoi chaque employé a passé un entretien individuel, abordant essentiellement le ressenti face au confinement et les conditions de télétravail. De plus, ma visite médicale obligatoire prévue initialement en avril a pu être passée en téléconsultation au cours du mois de mai.

Lors du déconfinement, l'entreprise Do&Go a une nouvelle fois été une oreille attentive aux craintes et aux préférences de chacun en proposant à ses employés de réintégrer les locaux ou bien de poursuivre le télétravail selon leur convenance. Des précautions sanitaires ont été mise en place : espacement des bureaux, installation de vitres anti-postillons, mise à disposition de gel hydroalcoolique et de masques de protection.

4.2 Planification des tâches

L'application « Gestion des extranet client » présentée dans ce rapport est une application représentant un service innovant de gestion des clients et des utilisateurs externes qui a été développé dans le but d'être fourni à plusieurs clients. Une première version fonctionnelle de l'application doit être créée pour la mi-juillet 2020. Ce projet étant une initiative propre à l'entreprise, il n'existe pas de « sprints » précis à proprement parler comportant des étapes de développement associées à des dates butoirs bien définies.

L'autonomie totale qui m'a été octroyée sur ce projet m'a donc permis de répartir les tâches de la manière suivante :

Dans un premier temps, concevoir l'application au niveau fonctionnel dans le studio Moovapps. Cela se traduit par la création des processus nécessaires, des étapes de workflow qui leur sont associées, et d'une table de configuration.

Dans un second temps, réaliser les algorithmes de génération d'identifiant et de génération de mot de passe.

Dans un troisième temps, développer les classes d'extensions utilisant ces algorithmes créés et nécessaires pour les lier à l'application.

Enfin, la réalisation de tests sera effectuée par le consultant fonctionnel et des nouvelles fonctionnalités seront ajoutées.

Les évolutions du code ont été gérées avec le logiciel GitKraken. Cela permet notamment une récupération des versions antérieures du code, un partage du code plus simple entre les collaborateurs du projet ou encore un suivi des temps entre les différents « commit ».

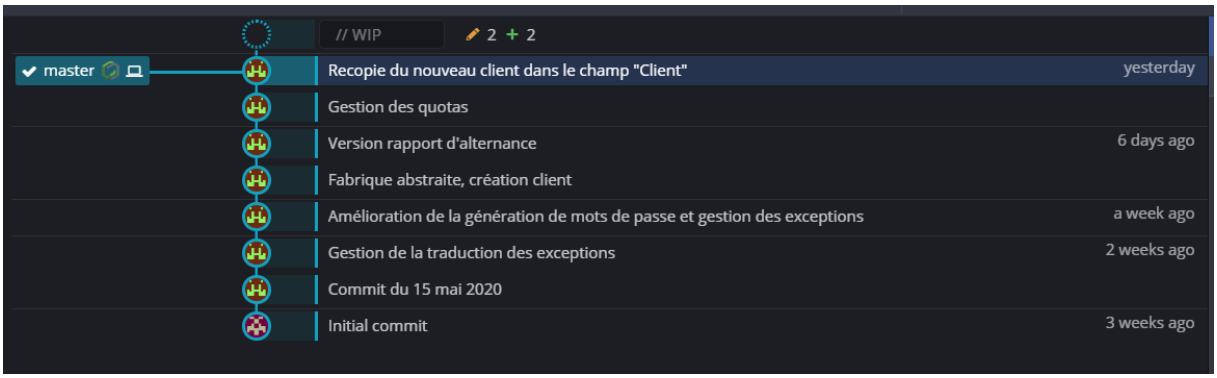

Figure 22 : Imprimé écran du logiciel GitKraken

4.3 Bilan critique par rapport au cahier des charges

Le projet est arrivé à son terme, non sans encombre. Le principal obstacle a été lié à la configuration de l'environnement de travail. En effet, une installation de Moovapps nécessite une connexion à une ou plusieurs bases de données. La version de Moovapps installée sur mon poste (15.5) supporte une base de données de type MySQL hébergée sur une plateforme de développement web, telle que WampServer. C'est la solution que j'ai choisi d'utiliser. Cependant, lorsque j'ai installé une version 16.4 de Moovapps en utilisant le même type d'hébergement de bases de données, il m'était impossible d'effectuer n'importe quelle opération nécessitant une interaction avec l'une des bases. La documentation d'installation de Moovapps 16.4 préconise l'utilisation d'un connecteur MySQL version 5.1.46, qui est différent de celui préconisé pour l'installation d'une version 15.5 (5.1.25). Après avoir pensé un temps que le problème venait de la version du connecteur MySQL utilisée et après avoir essayé une installation avec d'autres versions de connecteurs sans succès, j'ai installé une autre version de MySQL (1.5.29) que celle préconisée par la documentation d'installation (1.5.27) directement sur mon poste sans passer par WampServer. Tout fonctionnait normalement, le problème provenait donc d'une erreur de version de MySQL dans la documentation.

L’autre changement majeur entre la version 15 et la version 16 réside dans le fait que le serveur d’application utilisé n’est plus nommé JBOSS mais WILDFLY (bien que ce soit une version plus récente de JBOSS, WILDFLY possède une arborescence des dossiers différente). Cette version n’est pas prise en charge par le mode débug d’éclipse, il a donc fallu activer le mode débug en éditant un fichier de configuration puis démarrer le serveur web en ligne de commande. Cette opération engendre l’écriture des logs dans un terminal et non plus dans la console d’éclipse, ce qui a réduit considérablement le confort de développement par la suite.

Malgré tout, la conception a été réalisée en deux semaines, ce qui a laissé du temps pour se poser des questions pour rendre l’application encore plus « autonome », dans le but de faciliter le travail de déploiement du consultant fonctionnel. C’est pourquoi le packaging de l’application a été mis en place de manière à ce que le déploiement s’effectue de manière automatique. Des améliorations successives du code ont permis une meilleure organisation du système de l’application, une meilleure maintenabilité ainsi qu’une réduction du temps de traitement requêtes. Une ouverture possible sur une nouvelle fonctionnalité à intégrer à l’application est de créer une extension qui forcerait l’utilisateur à modifier son mot de passe lors de sa première connexion.

Conclusion

L'objectif de ce projet était de créer une application capable de gérer la création et le suivi de clients et d'utilisateurs.

La réalisation a été conforme aux besoins : l'utilisateur a le loisir de sélectionner un mode de génération d'identifiant parmi les trois méthodes proposées. Il peut également choisir le mode de génération du mot de passe selon ses motivations personnelles.

L'application fonctionne et le temps d'exécution du code est rapide. Cela est dû aux modifications successives de la méthode de génération du mot de passe qui ont contribué à améliorer la complexité de l'algorithme de cette méthode, et donc à optimiser son temps d'exécution. Le code est plus facilement maintenable en vue de futures éventuelles évolutions.

Les principales difficultés rencontrées au cours de la réalisation de ce projet ont finalement été annexes au développement de l'application en elle-même. En effet, c'est l'installation de la dernière version de Moovapps, nécessaire au bon fonctionnement de l'application « Extranet Client », qui a posé problème.

La création de la classe utilitaire a permis d'envisager une récupération des données plus rapidement et de manière plus organisée en vue d'un développement futur.

D'un point de vue technique, ce projet m'a permis d'acquérir de l'expérience sur le développement Java, mais également d'approfondir mes connaissances sur le déploiement et le fonctionnement d'une application web, les interactions client-serveur et le contenu des fichiers xml de configuration. J'ai également pu réaliser l'importance de l'amélioration de la complexité d'un algorithme. Au cours de ce projet, j'ai mis en œuvre de bonnes pratiques de développement vues tout au long de ma formation à l'IUT.

D'un point de vue humain, j'ai développé mon sens du travail en collectivité, de l'organisation, de la rigueur ainsi que de l'estimation des temps nécessaires à effectuer une ou plusieurs tâches.

Bibliographie

Martin, R. (2009). *Coder Proprement*. Edition Pearson.

Moovapps resources center (accès protégé) :
https://resourcescenter.moovapps.com/moovapps/easysite/workplace/custom/home

API Java :
https://docs.oracle.com/javase/7/docs/api

Vdoc kit (accès protégé) :
https://process-sdk.doc.moovapps.com/html/15.5/sdk

Javadoc Moovapps :
https://process-sdk.doc.moovapps.com/html/16.4/javadoc/

Annexes techniques

Annexe A : Code source de la classe « GenerateLoginAndPassword »

```
package com.doandgo.extranetclient.user;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import org.apache.commons.lang.NullArgumentException;

import com.axemble.vdoc.sdk.Modules;
import com.axemble.vdoc.sdk.document.extensions.BaseStorageResourceExtension;
import com.axemble.vdoc.sdk.interfaces.IProperty;
import com.axemble.vdoc.sdk.interfaces.IStorageResource;
import com.doandgo.commons.utils.StringUtils;
import com.doandgo.extranetclient.pojo.ConfigurationMotDePasse;
import com.doandgo.extranetclient.pojo.ConfigurationMotDePasseFactory;
import com.doandgo.extranetclient.pojo.ConfigurationMotDePasseFaibleFactory;
import com.doandgo.extranetclient.pojo.ConfigurationMotDePasseFortFactory;
import com.doandgo.extranetclient.user.IncoherentArgumentsException;
import com.doandgo.extranetclient.util.ExtranetClientUtil;
import com.doandgo.extranetclient.util.InvalidArgumentTypeException;
import com.doandgo.extranetclient.util.InvalidArgumentValueException;
import com.doandgo.extranetclient.util.NullParameterException;
import com.doandgo.extranetclient.util.NullValueReturnedByQueryException;
import com.doandgo.extranetclient.util.UserUtils;
import com.doandgo.moovapps.exceptions.VdocHelperException;
import com.doandgo.moovapps.utils.NullUtils;
import com.doandgo.moovapps.utils.VdocHelper;
import com.doandgo.moovapps.utils.exception.DataBaseConnectionException;

/**
 * Classe d'extension qui gère la génération d'un login au format (initiale du
 * prénom + nom) et d'un mot de passe aléatoire à la modification du bouton
 * radio "Méthode de gestion du login et du mot de passe"
 *
 * @author Thomas CHARMES
 */
public class GenerateLoginAndPassword extends BaseStorageResourceExtension {

	private static final long serialVersionUID = 1L;
	private static final Logger LOGGER =
Logger.getLogger(GenerateLoginAndPassword.class.getName());
	private static IStorageResource tableConfiguration = getTableConfiguration();

	@Override
	public void onPropertyChanged(IProperty property) {
		if
(property.getName().equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD)) {
			try {
				IStorageResource res = getStorageResource();
				String methodeGenerationSelectionnee = (String) res

	.getValue(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD);
				loadFieldsLoginFromMethod(res, methodeGenerationSelectionnee);
				loadFieldsPasswordFromMethod(res, methodeGenerationSelectionnee);
				manageEditability(res, methodeGenerationSelectionnee);
				res.save(getWorkflowModule().getSysadminContext());
			} catch (NullArgumentException e) {
				reportError("Argument null dans la méthode
loadFieldsLoginAndPasswordFromMethod : " + e.getMessage());
				e.printStackTrace();
			}
		}
		super.onPropertyChanged(property);
	}

	// Les champs login et mdp non éditables à l'ouverture du document si la méthode
	// de gestion est automatique
```

```
	@Override
	public boolean onAfterLoad() {
		IStorageResource res = getStorageResource();
		String methodeGenerationSelectionnee = (String) res
				.getValue(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD);
		manageEditability(res, methodeGenerationSelectionnee);
		return super.onAfterLoad();
	}

	public static IStorageResource getTableConfiguration() {
		Map<String, Object> equalsConstraints = new HashMap<String, Object>();
		IStorageResource tableConfiguration = null;
		try {
			tableConfiguration =
VdocHelper.getDataUniverseResource(Modules.getDirectoryModule(),
					Modules.getProjectModule(), Modules.getWorkflowModule(),
ExtranetClientUtil.ORGANIZATION_NAME,
					ExtranetClientUtil.APPLICATION_NAME,
ExtranetClientUtil.CATALOG_NAME, ExtranetClientUtil.TABLE_NAME,
					equalsConstraints);
		} catch (VdocHelperException e) {
			reportError("Erreur à la récupération de la table de configuration : " +
e.getMessage());
			e.printStackTrace();
		}
		if (tableConfiguration != null)
			return tableConfiguration;
		return null;
	}

	private void loadFieldsLoginFromMethod(IStorageResource res, String
methodeGenerationSelectionnee) throws NullArgumentException {
		String typeLogin = (String) tableConfiguration
				.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_TYPE_LOGIN);
		if (NullUtils.isNull(typeLogin)) {
			throw new NullArgumentException(typeLogin);
		}
		if
(methodeGenerationSelectionnee.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALU
E_AUTOMATIQUE)
				|| methodeGenerationSelectionnee

	.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_SEMI_AUTOMATIQUE)) {
			setLogin(res, typeLogin);
		} else {
			res.setValue(ExtranetClientUtil.TABLE_FIELD_LOGIN, null);
		}
	}

	private void setLogin(IStorageResource res, String typeLogin) {
		if
(typeLogin.equals(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_VALUE_NOM_TYPE_LOGIN))
			setLoginWithName(res);
		if
(typeLogin.equals(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_VALUE_EMAIL_TYPE_LOGIN))
			setLoginWithEmail(res);
		if
(typeLogin.equals(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_VALUE_SOCIETE_TYPE_LOGIN))
			try {
				setLoginWithSociete(res);
			} catch (InvalidArgumentValueException e) {
				reportError("La valeur d'au moins un des paramètres de la table de
configuration n'est pas autorisée : " + e.getMessage());
				e.printStackTrace();
			}
	}
```

```
	private void setLoginWithName(IStorageResource res) {
		String prenom = (String) res.getValue(ExtranetClientUtil.TABLE_FIELD_PRENOM);
		String nom = (String) res.getValue(ExtranetClientUtil.TABLE_FIELD_NOM);
		if (StringUtils.isEmpty(prenom) || StringUtils.isEmpty(nom)) {
			reportError("le prénom ou le nom est null");
			return;
		}
		try {
			res.setValue(ExtranetClientUtil.TABLE_FIELD_LOGIN,
UserUtils.generateLoginWithName(prenom, nom));
		} catch (NullParameterException | InvalidArgumentTypeException |
NullArgumentException e) {
			reportError("Erreur à la génération du login à partir du nom : " +
e.getMessage());
			e.printStackTrace();
		}
	}

	private void setLoginWithEmail(IStorageResource res) {
		String email = (String) res.getValue(ExtranetClientUtil.TABLE_FIELD_EMAIL);
		if (StringUtils.isEmpty(email)) {
			reportError("email null");
			return;
		}
		res.setValue(ExtranetClientUtil.TABLE_FIELD_LOGIN, email);
	}

	private void setLoginWithSociete(IStorageResource res) throws
InvalidArgumentValueException {
		IStorageResource tableConfiguration = getTableConfiguration();
		if (NullUtils.isNull(tableConfiguration)) {
			reportError("table de configuration null");
			return;
		}
		String societe = (String)
tableConfiguration.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_PREFIXE);
		if (StringUtils.isEmpty(societe)) {
			reportError("société null");
			return;
		}

		// if nbChiffresSuffixe = 4 et index départ = 1, premier login = 0001
		Float nbChiffresSuffixe = (Float)
tableConfiguration.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_CHIFFRES_SUFFIXE);
		Float indexDepart = (Float)
tableConfiguration.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_INDEX_DEPART);

		// Les vérifications sont effectuées dans la classe checkConfiguration

		try {
			res.setValue(ExtranetClientUtil.TABLE_FIELD_LOGIN,
UserUtils.generateLoginWithSociete(societe, indexDepart, nbChiffresSuffixe));
		} catch (NullArgumentException | InvalidArgumentTypeException |
DataBaseConnectionException | NullValueReturnedByQueryException e) {
			reportError("Erreur à la génération du login à partir de la société : " +
e.getMessage());
			e.printStackTrace();
		}
	}

	private void loadFieldsPasswordFromMethod(IStorageResource res, String
methodeGenerationSelectionnee) {
		String typePassword = (String) tableConfiguration
				.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_TYPE_PASSWORD);
		if (NullUtils.isNull(typePassword)) {
			throw new NullArgumentException(typePassword);
		}
```

```
            if (methodeGenerationSelectionnee

      .equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_AUTOMATIQUE)) {
                  setPassword(res, typePassword);
            } else {
                  res.setValue(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE, null);
                  res.setValue(ExtranetClientUtil.TABLE_FIELD_CONFIRMATION_MOT_DE_PASSE, null);
            }
      }

      private void setPassword(IStorageResource res, String typePassword) {
            Long nbNumbers = (Long)
tableConfiguration.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_NUMBERS);
            boolean donneesNumbersIncoherentes = (nbNumbers == null || nbNumbers < 0);
            if (donneesNumbersIncoherentes) {
                  reportError("Le nombre de chiffres n'est pas correctement défini");
                  return;
            }
            if
(typePassword.equals(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_TYPE_PASSWORD_VALUE_WEAK)) {
                  setWeakPassword(res, nbNumbers);
            }
            if
(typePassword.equals(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_TYPE_PASSWORD_VALUE_STRONG))
{
                  setStrongPassword(res, nbNumbers);
            }
      }

      private void setWeakPassword(IStorageResource res, Long nbNumbers) {
            Long nbCharsTotal = (Long) tableConfiguration
                        .getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_TOTAL_CHARS);
            boolean donneesCharTotalIncoherentes = (nbCharsTotal == null || nbCharsTotal < 0);
            if (donneesCharTotalIncoherentes) {
                  reportError("Le nombre de caractère total n'est pas correctement défini");
                  return;
            }
            try {
                  ConfigurationMotDePasse configFaible = ConfigurationMotDePasseFactory
                              .getConfigurationMotDePasse(new
ConfigurationMotDePasseFaibleFactory(nbCharsTotal, nbNumbers));
                  res.setValue(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE,
                              UserUtils.generateRandomWeakPassword(configFaible));
            } catch (IncoherentArgumentsException e) {
                  reportError("erreur à la génération du mot de passe faible : " +
e.getMessage());
                  e.printStackTrace();
            }
      }

      private void setStrongPassword(IStorageResource res, Long nbNumbers) {
            Long nbSpecialChars = (Long) tableConfiguration

      .getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_SPECIAL_CHARS);
            Long nbMajuscules = (Long) tableConfiguration
                        .getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_MAJUSCULES);
            Long nbMinuscules = (Long) tableConfiguration
                        .getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_MINUSCULES);
            ConfigurationMotDePasse configForte =
ConfigurationMotDePasseFactory.getConfigurationMotDePasse(
                        new ConfigurationMotDePasseFortFactory(nbMajuscules, nbMinuscules,
nbSpecialChars, nbNumbers));
            res.setValue(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE,
UserUtils.generateRandomStrongPassword(configForte));
      }
```

```
	private void manageEditability(IStorageResource res, String
methodeGenerationSelectionnee) {
		if (methodeGenerationSelectionnee

	.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_AUTOMATIQUE)) {
			getResourceController().setEditable(ExtranetClientUtil.TABLE_FIELD_LOGIN,
false);

	getResourceController().setEditable(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE, false);
		}
		if (methodeGenerationSelectionnee

	.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_SEMI_AUTOMATIQUE)) {
			getResourceController().setEditable(ExtranetClientUtil.TABLE_FIELD_LOGIN,
false);

	getResourceController().setEditable(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE, true);
		}
		if
(methodeGenerationSelectionnee.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALU
E_MANUEL)) {
			getResourceController().setEditable(ExtranetClientUtil.TABLE_FIELD_LOGIN,
true);

	getResourceController().setEditable(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE, true);
		}
	}

	private static void reportError(String log) {
		LOGGER.info(log);
		System.out.println(log);
	}
}
```

Annexe B : Code source de la classe « UserUtils »

```java
package com.doandgo.extranetclient.util;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Random;

import org.apache.commons.lang.NullArgumentException;

import com.axemble.vdoc.sdk.Modules;
import com.axemble.vdoc.sdk.document.extensions.BaseDocumentExtension;
import com.axemble.vdoc.sdk.interfaces.IUser;
import com.axemble.vdoc.sdk.modules.IDirectoryModule;
import com.axemble.vdoc.sdk.modules.IWorkflowModule;
import com.doandgo.commons.utils.RegexpUtils;
import com.doandgo.commons.utils.StringUtils;
import com.doandgo.extranetclient.pojo.ConfigurationMotDePasse;
import com.doandgo.extranetclient.user.IncoherentArgumentsException;
import com.doandgo.moovapps.utils.DataSourceCenter;
import com.doandgo.moovapps.utils.NullUtils;
import com.doandgo.moovapps.utils.SQLUtils;
import com.doandgo.moovapps.utils.exception.DataBaseConnectionException;

/**
 * Gère la randomisation des mots de passe et la création d'un login
 *
 * @author Thomas CHARMES
 */
public class UserUtils extends BaseDocumentExtension {
	private static final long serialVersionUID = 1L;

	/**
	 * crée un login de la forme initiale du prénom + nom sans les caractères
	 * spéciaux
	 *
	 * @param prenom
	 * @param nom
	 * @return
	 * @throws NullArgumentException
	 * @throws NullParameterException
	 * @throws InvalidArgumentTypeException
	 */
	public static String generateLoginWithName(String prenom, String nom)
			throws NullArgumentException, NullParameterException,
InvalidArgumentTypeException {

		if (NullUtils.isAtLeastOneNullOrEmpty(prenom, nom))
			throw new NullParameterException("Le nom ou le prénom est une chaîne de
caractères nulle ou vide");

		prenom = prenom.trim().toLowerCase();
		nom = nom.trim().toLowerCase();

		if (!RegexpUtils.isStartByLowercase(prenom) || RegexpUtils.hasNumber(nom) ||
RegexpUtils.hasSpecialCharacter(nom))
			throw new InvalidArgumentTypeException(
					"Le nom ou l'initiale du prénom comporte au moins un chiffre ou
un caractère spécial");

		String login = null;
		Character initialeDuPrenom = prenom.charAt(0);
		String[] tableauNom = nom.split("");
```

```
            tableauNom = replaceSpecialCharsByLetter(tableauNom);
            String nomSansAccent = tableauToString(tableauNom);
            login = initialeDuPrenom + nomSansAccent;
            try {
                  login = incrementeLeLoginSiDejaExistant(login);
            } catch (DataBaseConnectionException | NullValueReturnedByQueryException e) {
                  reportError("Erreur lors de l'incrémentation du login : " + e.getMessage());
                  e.printStackTrace();
            }
            return login;
      }

      private static String[] replaceSpecialCharsByLetter(String[] tableauNom) {
            for (int i = 0; i < tableauNom.length; i++) {
                  if (hasSpecialChar(tableauNom[i])) {
                        tableauNom[i] = modifySpecialCharacter(tableauNom[i]);
                  }
            }
            return tableauNom;
      }

      /**
       * booléen qui renvoie true si la lettre passée en paramètres est un caractère
       * spécial renvoie false sinon
       *
       * @param lettre
       * @return
       */
      private static boolean hasSpecialChar(String lettre) {
            List<String> listSpecialCharacters = getListSpecialCharacters();
            boolean isSpecialCharacter = false;
            for (String element : listSpecialCharacters) {
                  if (lettre.equals(element))
                        isSpecialCharacter = true;
            }
            return isSpecialCharacter;
      }

      /**
       * Remplace une lettre avec accent par la même lettre sans accent supprime le
       * "-" dans un nom composé par exemple
       *
       * @param lettre
       * @return
       */
      private static String modifySpecialCharacter(String lettre) {
            if (!lettre.equals(null)) {
                  String nouvelleLettre = null;
                  if (lettre.equals("é") || lettre.equals("è") || lettre.equals("ê") ||
lettre.equals("ë"))
                        nouvelleLettre = "e";
                  if (lettre.equals("ä") || lettre.equals("à") || lettre.equals("â"))
                        nouvelleLettre = "a";
                  if (lettre.equals("ï") || lettre.equals("î"))
                        nouvelleLettre = "i";
                  if (lettre.equals("ö") || lettre.equals("ô"))
                        nouvelleLettre = "o";
                  if (lettre.equals("ù") || lettre.equals("ü") || lettre.equals("û"))
                        nouvelleLettre = "u";
                  if (lettre.equals("ç"))
                        nouvelleLettre = "c";
                  if (lettre.equals("-"))
                        nouvelleLettre = "";
                  return nouvelleLettre;
            }
            return null;
      }
```

```
	private static String tableauToString(String[] tableauNom) {
		String nomSansAccent = "";
		for (int i = 0; i < tableauNom.length; i++) {
			if (!StringUtils.isEmpty(tableauNom[i])) {
				nomSansAccent = nomSansAccent + tableauNom[i];
			}
		}
		return nomSansAccent.trim();
	}

	private static String incrementeLeLoginSiDejaExistant(String loginInitial)
			throws DataBaseConnectionException, NullValueReturnedByQueryException {
		int i = 1;
		String login = loginInitial;
		while (alreadyExistsInDirectoryModule(login)) {
			// while (alreadyExistsInDirectoryModule(login)) {
			login = loginInitial + Integer.toString(i);
			i++;
		}
		return login;
	}

	public static boolean alreadyExistsInDirectoryModule(String login) {
		IDirectoryModule dm = null;
		try {
			dm = Modules.getDirectoryModule();
			IUser userByLogin = dm.getUserByLogin(login);
			if (! NullUtils.isNull(userByLogin))
				return true;
		}
		finally {
			Modules.releaseModule(dm);
		}
		return false;
	}

	private static boolean alreadyExistsInDatabase(String login)
			throws DataBaseConnectionException, NullValueReturnedByQueryException {
		boolean exist = false;
		String query = "SELECT login FROM vdoc_user";
		Connection externalConnection = DataSourceCenter
				.getExternalConnection(ExtranetClientUtil.MOOVAPPS_EXTERNAL_CONNEXION);
		ResultSet rs = null;
		try {
			rs =
SQLUtils.establishConnectiontoDatabaseAndExecuteSimpleQuery(externalConnection, query);
			if (NullUtils.isNull(rs))
				throw new DataBaseConnectionException(
						"Echec lors de la connexion à la base de données et de
l'exécution de la requête");
			String loginDatabase = "";
			while (rs.next()) {
				loginDatabase = rs.getString("login");
				if (rs.wasNull()) {
					throw new NullValueReturnedByQueryException("La requête n'a
renvoyé aucune valeur");
				}
				if (loginDatabase.equals(login)) {
					exist = true;
				}
			}
			rs.close();
		} catch (DataBaseConnectionException | SQLException e) {
			reportError("Erreur lors de la tentative de connexion à la base de données et
d'éxécution de la requête : "
					+ e.getMessage());
```

```
		}
		return exist;
	}

	public static String generateLoginWithSociete(String prefixeSociete, Float indexDepart,
Float nbChiffresSuffixe)
			throws NullArgumentException, InvalidArgumentTypeException,
DataBaseConnectionException, NullValueReturnedByQueryException, InvalidArgumentValueException {

		if (NullUtils.isNullOrEmpty(prefixeSociete))
			throw new InvalidArgumentTypeException("La société est vide");

		if (! indexContientMoinsDeChiffresQueLeSuffixe(indexDepart.intValue(),
nbChiffresSuffixe.intValue())) {
			throw new InvalidArgumentValueException("L'index de départ contient plus de
chiffres que le suffixe souhaité");
		}

		int i = indexDepart.intValue();
		String login = prefixeSociete + "-";
		String suffixe = StringUtils.cutAndFill(Integer.toString(i),
nbChiffresSuffixe.intValue(), true, "0", false);

		while (alreadyExistsInDatabase(login + suffixe)) {
			i++;
			suffixe = StringUtils.cutAndFill(Integer.toString(i),
nbChiffresSuffixe.intValue(), true, "0", false);
		}

		if (! indexContientMoinsDeChiffresQueLeSuffixe(i, nbChiffresSuffixe.intValue())) {
			throw new InvalidArgumentValueException("L'index contient plus de chiffres
que le suffixe souhaité");
		}

		return login + suffixe;
	}

	public static boolean indexContientMoinsDeChiffresQueLeSuffixe(int index, int
nbChiffresSuffixe) throws InvalidArgumentValueException {

		if (index < 10)
			return true;
		if (index < 100) {
			if (nbChiffresSuffixe < 2)
				return false;
			else return true;
		}
		if (index < 1000) {
			if (nbChiffresSuffixe < 3)
				return false;
			else return true;
		}
		if (index < 10000) {
			if (nbChiffresSuffixe < 4)
				return false;
			else return true;
		}
		if (index < 100000) {
			if (nbChiffresSuffixe < 5)
				return false;
			else return true;
		}
		return false;
	}

	public static String generateRandomStrongPassword(ConfigurationMotDePasse config) {
		// Récupération des éléments de la configuration :
```

```
            Long nbMajLeft = config.getNbMajuscules();
            Long nbMinLeft = config.getNbMinuscules();
            Long nbNumbersLeft = config.getNbNumbers();
            Long nbSpecialCharsLeft = config.getNbSpecialChars();
            Long nbCharactersLeft = config.getNbTotalChars();
            String password = "";
            String lettre = "";
            Map<String, Long> listOfAllParameters = reinitializeListStrongParameters(nbMajLeft,
nbMinLeft, nbNumbersLeft,
                        nbSpecialCharsLeft);

            while (nbCharactersLeft > 0) {
                  Map<String, Long> listOfParametersLeft =
getParametersLeft(listOfAllParameters);
                  List<String> listOfAuthorizedCharacters =
getListOfAuthorizedCharacters(listOfParametersLeft);
                  lettre = getRandomCharacterIn(listOfAuthorizedCharacters);
                  password += lettre;
                  listOfAllParameters = decrementStrongType(lettre, nbMajLeft, nbMinLeft,
nbNumbersLeft, nbSpecialCharsLeft);
                  for (Entry<String, Long> param : listOfAllParameters.entrySet()) {
                        if (param.getKey().equals("nbMajLeft"))
                              nbMajLeft = param.getValue();
                        if (param.getKey().equals("nbMinLeft"))
                              nbMinLeft = param.getValue();
                        if (param.getKey().equals("nbNumbersLeft"))
                              nbNumbersLeft = param.getValue();
                        if (param.getKey().equals("nbSpecialCharsLeft"))
                              nbSpecialCharsLeft = param.getValue();
                  }
                  nbCharactersLeft--;
            }
            return password;
      }

      /**
       * Renvoie une map (clé=valeur) de tous les paramètres mis à jour après chaque
       * tirage
       *
       * @param nbMajLeft
       * @param nbMinLeft
       * @param nbNumbersLeft
       * @param nbSpecialCharsLeft
       * @return
       */
      private static Map<String, Long> reinitializeListStrongParameters(Long nbMajLeft, Long
nbMinLeft,
                  Long nbNumbersLeft, Long nbSpecialCharsLeft) {
            Map<String, Long> parameters = new HashMap<String, Long>();
            parameters.put("nbMajLeft", nbMajLeft);
            parameters.put("nbMinLeft", nbMinLeft);
            parameters.put("nbNumbersLeft", nbNumbersLeft);
            parameters.put("nbSpecialCharsLeft", nbSpecialCharsLeft);
            return parameters;
      }

      private static Map<String, Long> reinitializeListWeakParameters(Long nbMinLeft, Long
nbNumbersLeft) {
            Map<String, Long> parameters = new HashMap<String, Long>();
            parameters.put("nbMinLeft", nbMinLeft);
            parameters.put("nbNumbersLeft", nbNumbersLeft);
            return parameters;
      }

      /**
       * renvoie une map de tous les paramètres non-nuls
       *
```

```
	 * @param listParameter
	 * @return
	 */
	private static Map<String, Long> getParametersLeft(Map<String, Long> listParameters) {
		Map<String, Long> listParametersLeft = new HashMap<String, Long>();
		for (Entry<String, Long> param : listParameters.entrySet()) {
			if (param.getValue() > 0L)
				listParametersLeft.put(param.getKey(), param.getValue());
		}
		return listParametersLeft;
	}

	/**
	 * Renvoie la liste des caractères autorisés en fonction des paramètres restants
	 *
	 * @param listOfParametersLeft
	 * @return
	 */
	private static List<String> getListOfAuthorizedCharacters(Map<String, Long>
listOfParametersLeft) {
		List<String> listOfPossibleCharacters = new ArrayList<String>();
		for (Entry<String, Long> param : listOfParametersLeft.entrySet()) {
			if (param.getKey().equals("nbMajLeft")) {
				listOfPossibleCharacters.addAll(getListLettersMajuscules());
			}
			if (param.getKey().equals("nbMinLeft")) {
				listOfPossibleCharacters.addAll(getListLettersMinuscules());
			}
			if (param.getKey().equals("nbNumbersLeft")) {
				listOfPossibleCharacters.addAll(getListNumbers());
			}
			if (param.getKey().equals("nbSpecialCharsLeft")) {
				listOfPossibleCharacters.addAll(getListSpecialCharacters());
			}
		}
		return listOfPossibleCharacters;
	}

	private static List<String> getListSpecialCharacters() {
		List<String> liste = new ArrayList<String>();
		fillListWithOneStringCharacter(liste, "éèçàùêëöâîïäôü-
&\"\\'~²#{([|,^@)]°+=}/*.!§:;?<>");
		return liste;
	}

	private static List<String> getListNumbers() {
		List<String> liste = new ArrayList<String>();
		fillListWithOneStringCharacter(liste, "0123456789");
		return liste;
	}

	private static List<String> getListLettersMinuscules() {
		List<String> liste = new ArrayList<String>();
		fillListWithOneStringCharacter(liste, "abcdefghijklmnopqrstuvwxyz");
		return liste;
	}

	private static List<String> getListLettersMajuscules() {
		List<String> liste = new ArrayList<String>();
		List<String> listLettersMinuscules = getListLettersMinuscules();
		for (String lettreMinuscule : listLettersMinuscules) {
			liste.add(lettreMinuscule.toUpperCase());
		}
		return liste;
	}

	/**
```

```
	 * Rempli la liste avec chaque caractère de la chaine de caractère passée en
	 * paramètres
	 *
	 * @param liste
	 * @param stringToFill
	 */
	private static void fillListWithOneStringCharacter(List<String> liste, String
stringToFill) {
		if (liste != null && !StringUtils.isEmpty(stringToFill)) {
			String[] tableauStringToFill = stringToFill.trim().split("");
			for (int i = 0; i < tableauStringToFill.length; i++) {
				if (!StringUtils.isEmpty(tableauStringToFill[i]))
					liste.add(tableauStringToFill[i]);
			}
		}
	}

	private static String getRandomCharacterIn(List<String> list) {
		Random random = new Random();
		String lettre;
		int randomInt = random.nextInt(list.size());
		lettre = list.get(randomInt);
		return lettre;
	}

	/**
	 * Décrémente la valeur du nombre de tirages restants du type correspondant à la
	 * lettre
	 */
	private static Map<String, Long> decrementStrongType(String lettre, Long nbMajLeft, Long
nbMinLeft,
			Long nbNumbersLeft, Long nbSpecialCharsLeft) {
		if (getListLettersMajuscules().contains(lettre))
			nbMajLeft--;
		if (getListLettersMinuscules().contains(lettre))
			nbMinLeft--;
		if (getListNumbers().contains(lettre))
			nbNumbersLeft--;
		if (getListSpecialCharacters().contains(lettre))
			nbSpecialCharsLeft--;
		return reinitializeListStrongParameters(nbMajLeft, nbMinLeft, nbNumbersLeft,
nbSpecialCharsLeft);
	}

	public static String generateRandomWeakPassword(ConfigurationMotDePasse config)
			throws IncoherentArgumentsException {
		IWorkflowModule wm = Modules.getWorkflowModule();
		Long nbNumbersLeft = config.getNbNumbers();
		Long nbCharactersLeft = config.getNbTotalChars();
		Long nbMinLeft = nbCharactersLeft - nbNumbersLeft - 1;
		// Les paramètres sont pertinents
		if (nbCharactersLeft <= nbNumbersLeft) {
			if (wm == null)
				throw new IncoherentArgumentsException("Incohérence des arguments !");
			else {
				throw new
IncoherentArgumentsException(wm.getStaticString("LG_TRAD_INCOHERENT_ARGS"));
			}
		}
		String password = "";
		String lettre = "";
		// Attribution de la majuscule initiale :
		lettre = getRandomCharacterIn(getListLettersMajuscules());
		password = lettre;
		nbCharactersLeft--;
```

```
            Map<String, Long> listOfAllParameters = reinitializeListWeakParameters(nbMinLeft,
nbNumbersLeft);
            while (nbCharactersLeft > 0) {
                  Map<String, Long> listOfParametersLeft =
getParametersLeft(listOfAllParameters);
                  List<String> listOfAuthorizedCharacters =
getListOfAuthorizedCharacters(listOfParametersLeft);
                  lettre = getRandomCharacterIn(listOfAuthorizedCharacters);
                  password += lettre;
                  listOfAllParameters = decrementWeakType(lettre, nbMinLeft, nbNumbersLeft);
                  for (Entry<String, Long> param : listOfAllParameters.entrySet()) {
                        if (param.getKey().equals("nbMinLeft"))
                              nbMinLeft = param.getValue();
                        if (param.getKey().equals("nbNumbersLeft"))
                              nbNumbersLeft = param.getValue();
                  }
                  nbCharactersLeft--;
            }
            return password;
      }

      private static Map<String, Long> decrementWeakType(String lettre, Long nbMinLeft, Long
nbNumbersLeft) {
            if (getListLettersMinuscules().contains(lettre))
                  nbMinLeft--;
            if (getListNumbers().contains(lettre))
                  nbNumbersLeft--;
            return reinitializeListWeakParameters(nbMinLeft, nbNumbersLeft);
      }

      private static void reportError(String log) {
            LOGGER.info(log);
            System.out.println(log);
      }
```

Annexe C : Ancienne version de l’algorithme de génération de mot de passe sécurisé « generateRandomStrongPassword(ConfigurationMotDePasse config) »

```java
/**
 * génère un mot de passe aléatoire en tenant compte des paramètres fournis
 * On ne peut tirer que parmis les paramètres souhaités
 *
 * @param nbCharsTotal
 * @param nbMajuscules
 * @param nbMinuscules
 * @param nbNumbers
 * @param nbSpecialChars
 * @param prefixe
 * @return
 * @throws IncoherentArgumentsException
 */
public static String generateRandomStrongPassword(ConfigurationMotDePasseFort config)
            throws IncoherentArgumentsException {

      // Récupération des éléments de la configuration :
      Long nbMajuscules = config.getNbMajuscules();
      Long nbMinuscules = config.getNbMinuscules();
      Long nbNumbers = config.getNbNumbers();
      Long nbSpecialChars = config.getNbSpecialChars();
      Long nbCharsTotal = nbMajuscules + nbMinuscules + nbNumbers + nbSpecialChars;

      if (nbCharsTotal != (nbMajuscules + nbMinuscules + nbNumbers + nbSpecialChars) && nbCharsTotal !=
0)
            throw new IncoherentArgumentsException(
                        "Le nombre total de caractères n'est pas cohérent avec la configuration
choisie");

      // Paramètres originaux :
      Long nbCharactersLeft = nbCharsTotal;
      Long nbNumbersLeft = nbNumbers;
      Long nbMajLeft = nbMajuscules;
      Long nbMinLeft = nbMinuscules;
      Long nbSpecialCharsLeft = nbSpecialChars;
      String password = "";

      // Récupération des listes :
      // Listes avec 1 élément :
      List<String> listLettersMajuscules = getListLettersMajuscules();
      List<String> listLettersMinuscules = getListLettersMinuscules();
      List<String> listSpecialCharacters = getListSpecialCharacters();
      List<String> listNumbers = getListNumbers();

      // Listes avec 2 éléments :
      List<String> listOfSpecialsAndNumbers = getListOfSpecialsAndNumbers();
      List<String> listOfSpecialsAndMajuscules = getListOfSpecialsAndMajuscules();
      List<String> listOfSpecialsAndMinuscules = getListOfSpecialsAndMinuscules();
      List<String> listOfMajusculesAndNumbers = getListOfMajusculesAndNumbers();
      List<String> listOfMinusculesAndNumbers = getListOfMinusculesAndNumbers();
      List<String> listOfMajusculesAndMinuscules = getListOfMajusculesAndMinuscules();

      // Listes avec 3 éléments :
      List<String> listOfAllCharactersExceptSpecials = getListOfAllCharactersExceptSpecials();
      List<String> listOfAllCharactersExceptMajuscules = getListOfAllCharactersExceptMajuscules();
      List<String> listOfAllCharactersExceptMinuscules = getListOfAllCharactersExceptMinuscules();
      List<String> listOfAllCharactersExceptNumbers = getListOfAllCharactersExceptNumbers();

      // Liste avec 4 éléments :
      List<String> listOfAllCharacters = getListOfAllCharacters();

      String lettre = "";

      while (nbCharactersLeft > 0) {

            // Si on peut encore tout tirer
            if (nbMajLeft > 0 && nbMinLeft > 0 && nbSpecialCharsLeft > 0 && nbNumbersLeft > 0) {

                  // Tirage d'une lettre parmi toutes les possibilités :
                  lettre = getRandomCharacterIn(listOfAllCharacters);

                  password = password + lettre;
                  nbCharactersLeft--;

                  if (nbCharactersLeft == 0)
```

```
                        return password;

                  // Vérif type de la lettre tirée
                  if (listLettersMajuscules.contains(lettre)) {
                        nbMajLeft--;
                  } else {
                        if (listLettersMinuscules.contains(lettre)) {
                              nbMinLeft--;
                        } else {
                              if (listNumbers.contains(lettre)) {
                                    nbNumbersLeft--;
                              } else {
                                    if (listSpecialCharacters.contains(lettre)) {
                                          nbSpecialCharsLeft--;
                                    }
                              }
                        }
                  }
            }
            // On ne peut pas tout tirer
            else {
                  // On peut tirer tout sauf un caractère spécial
                  if (nbMajLeft > 0 && nbMinLeft > 0 && nbSpecialCharsLeft == 0 && nbNumbersLeft > 0) {

                        // Tirage d'une lettre parmi toutes les possibilités sauf le caractère spécial
:
                        lettre = getRandomCharacterIn(listOfAllCharactersExceptSpecials);
                        password = password + lettre;
                        nbCharactersLeft--;

                        if (nbCharactersLeft == 0)
                              return password;

                        // Vérif type de la lettre tirée
                        if (listLettersMajuscules.contains(lettre)) {
                              nbMajLeft--;
                        } else {
                              if (listLettersMinuscules.contains(lettre)) {
                                    nbMinLeft--;
                              } else {
                                    if (listNumbers.contains(lettre)) {
                                          nbNumbersLeft--;
                                    }
                              }
                        }

                  } else {

                        // Si on peut tout tirer sauf une majuscule
                        if (nbMajLeft == 0 && nbMinLeft > 0 && nbSpecialCharsLeft > 0 && nbNumbersLeft
> 0) {
                              // Tirage d'une lettre parmi toutes les possibilités sauf la majuscule :
                              lettre = getRandomCharacterIn(listOfAllCharactersExceptMajuscules);
                              password = password + lettre;
                              nbCharactersLeft--;

                              if (nbCharactersLeft == 0)
                                    return password;

                              // Vérif type de la lettre tirée
                              if (listLettersMinuscules.contains(lettre)) {
                                    nbMinLeft--;
                              } else {
                                    if (listNumbers.contains(lettre)) {
                                          nbNumbersLeft--;
                                    } else {
                                          if (listSpecialCharacters.contains(lettre)) {
                                                nbSpecialCharsLeft--;
                                          }
                                    }
                              }
                        } else {
                              // Si on peut tout tirer sauf une minuscule
                              if (nbMajLeft > 0 && nbMinLeft == 0 && nbSpecialCharsLeft > 0 &&
nbNumbersLeft > 0) {
```

```
                                        // Tirage d'une lettre parmi toutes les possibilités sauf la
minuscule :
                                        lettre =
getRandomCharacterIn(listOfAllCharactersExceptMinuscules);
                                        password = password + lettre;
                                        nbCharactersLeft--;

                                        if (nbCharactersLeft == 0)
                                                return password;

                                        // Vérif type de la lettre tirée
                                        if (listLettersMajuscules.contains(lettre)) {
                                                nbMajLeft--;
                                        } else {
                                                if (listNumbers.contains(lettre)) {
                                                        nbNumbersLeft--;
                                                } else {
                                                        if (listSpecialCharacters.contains(lettre)) {
                                                                nbSpecialCharsLeft--;
                                                        }
                                                }
                                        }
                                } else {
                                        // Si on peut tout tirer sauf un nombre
                                        if (nbMajLeft > 0 && nbMinLeft > 0 && nbSpecialCharsLeft > 0 &&
nbNumbersLeft == 0) {
                                                // Tirage d'une lettre parmi toutes les possibilités sauf
le nombre :
                                                lettre =
getRandomCharacterIn(listOfAllCharactersExceptNumbers);
                                                password = password + lettre;
                                                nbCharactersLeft--;

                                                if (nbCharactersLeft == 0)
                                                        return password;

                                                // Vérif type de la lettre tirée
                                                if (listLettersMajuscules.contains(lettre)) {
                                                        nbMajLeft--;
                                                } else {
                                                        if (listLettersMinuscules.contains(lettre)) {
                                                                nbMinLeft--;
                                                        } else {
                                                                if (listSpecialCharacters.contains(lettre)) {
                                                                        nbSpecialCharsLeft--;
                                                                }
                                                        }
                                                }
                                        } else {
                                                // Si on ne peut tirer qu'un caractère spécial
                                                if (nbMajLeft == 0 && nbMinLeft == 0 && nbSpecialCharsLeft
> 0 && nbNumbersLeft == 0) {
                                                        // Tirage d'une lettre parmi les caractères spéciaux
:
                                                        lettre = getRandomCharacterIn(listSpecialCharacters);
                                                        password = password + lettre;
                                                        nbCharactersLeft--;
                                                        nbSpecialCharsLeft--;

                                                        if (nbCharactersLeft == 0)
                                                                return password;
                                                } else {
                                                        // Si on ne peut tirer qu'un chiffre
                                                        if (nbMajLeft == 0 && nbMinLeft == 0 &&
nbSpecialCharsLeft == 0
                                                                        && nbNumbersLeft > 0) {
                                                                // Tirage d'une lettre parmi les chiffres :
                                                                lettre = getRandomCharacterIn(listNumbers);
                                                                password = password + lettre;
                                                                nbCharactersLeft--;
                                                                nbNumbersLeft--;

                                                                if (nbCharactersLeft == 0)
                                                                        return password;
                                                        } else {
```

```
nbSpecialCharsLeft == 0

majuscules :

getRandomCharacterIn(listLettersMajuscules);




nbSpecialCharsLeft == 0

minuscules :

getRandomCharacterIn(listLettersMinuscules);




majuscule

0 && nbSpecialCharsLeft > 0

0) {

parmi toutes les possibilités sauf la

getRandomCharacterIn(listOfSpecialsAndMajuscules);

lettre;




tirée

(listLettersMajuscules.contains(lettre)) {


(listSpecialCharacters.contains(lettre)) {

	nbSpecialCharsLeft--;



spécial et une minuscule

nbMinLeft > 0 && nbSpecialCharsLeft > 0

nbNumbersLeft == 0) {

lettre parmi toutes les possibilités :

getRandomCharacterIn(listOfSpecialsAndMinuscules);

+ lettre;
```

```
// Si on ne peut tirer qu'une majuscule
if (nbMajLeft > 0 && nbMinLeft == 0 &&

		&& nbNumbersLeft == 0) {
	// Tirage d'une lettre parmi les

	lettre =

	password = password + lettre;
	nbCharactersLeft--;
	nbMajLeft--;

	if (nbCharactersLeft == 0)
		return password;
} else {
	// Si on ne peut tirer qu'une minuscule
	if (nbMajLeft == 0 && nbMinLeft > 0 &&

			&& nbNumbersLeft == 0) {
		// Tirage d'une lettre parmi les

		lettre =

		password = password + lettre;
		nbCharactersLeft--;
		nbMinLeft--;

		if (nbCharactersLeft == 0)
			return password;
		} else {
			// Si on peut tirer un spé et une

			if (nbMajLeft > 0 && nbMinLeft ==

					&& nbNumbersLeft ==

				// Tirage d'une lettre

				// minuscule :
				lettre =

				password = password +

				nbCharactersLeft--;

				if (nbCharactersLeft == 0)
					return password;

				// Vérif type de la lettre

				if

					nbMajLeft--;
				} else {
					if



					}
				}
			} else {
				// Si on peut tirer un

				if (nbMajLeft == 0 &&

						&&

					// Tirage d'une

					lettre =

					password = password

					nbCharactersLeft--;
```

```
== 0)
password;

lettre tirée
(listLettersMinuscules.contains(lettre)) {

(listSpecialCharacters.contains(lettre)) {
	nbSpecialCharsLeft--;


un spé et un nombre
&& nbMinLeft == 0 && nbSpecialCharsLeft > 0
nbNumbersLeft > 0) {
d'une lettre parmi toutes les possibilités :
getRandomCharacterIn(listOfSpecialsAndNumbers);
password + lettre;
	nbCharactersLeft--;

(nbCharactersLeft == 0)
password;

de la lettre tirée
(listNumbers.contains(lettre)) {
	nbNumbersLeft--;

(listSpecialCharacters.contains(lettre)) {
	nbSpecialCharsLeft--;


tirer une majuscule et un nombre
> 0 && nbMinLeft == 0
	&& nbSpecialCharsLeft == 0 && nbNumbersLeft > 0) {
Tirage d'une lettre parmi toutes les possibilités
= getRandomCharacterIn(
	listOfMajusculesAndNumbers);
	password = password + lettre;
	nbCharactersLeft--;

(nbCharactersLeft == 0)
	return password;
```

```
if (nbCharactersLeft
	return

// Vérif type de la
if
	nbMinLeft--;
} else {
	if


	}
	}
} else {
	// Si on peut tirer
	if (nbMajLeft == 0
		&&
	// Tirage
	lettre =
	password =


	if
		return

	// Vérif type
	if


	} else {
		if


		}
	}

} else {
	// Si on peut
	if (nbMajLeft


		//
		// :
		lettre


		if
```

```
                                                                                //
Vérif type de la lettre tirée
                                                                                if
(listLettersMajuscules.contains(lettre)) {
        nbMajLeft--;
                                                                                } else
{

        if (listNumbers.contains(lettre)) {
        nbNumbersLeft--;
        }
                                                                                }
                                                                        } else {
                                                                                // Si
on peut tirer une minuscule et un nombre
                                                                                if
(nbMajLeft == 0 && nbMinLeft > 0
        && nbSpecialCharsLeft == 0
        && nbNumbersLeft > 0) {
        // Tirage d'une lettre parmi toutes les
        // possibilités :
        lettre = getRandomCharacterIn(
                listOfMinusculesAndNumbers);
        password = password + lettre;
        nbCharactersLeft--;

        if (nbCharactersLeft == 0)
        return password;

        // Vérif type de la lettre tirée

        if (listLettersMinuscules.contains(lettre)) {
        nbMinLeft--;
        } else {
        if (listNumbers.contains(lettre)) {
                nbNumbersLeft--;
        }
        }
                                                                                } else
{
        // Si on peut tirer une majuscule et une
        // minuscule
        if (nbMajLeft > 0 && nbMinLeft > 0
                && nbSpecialCharsLeft == 0
                && nbNumbersLeft == 0) {
        // Tirage d'une lettre parmi toutes les
```

```
        // possibilités :
        lettre = getRandomCharacterIn(
                    listOfMajusculesAndMinuscules);
        password = password + lettre;
        nbCharactersLeft--;

        if (nbCharactersLeft == 0)
              return password;

        // Vérif type de la lettre tirée
        if (listLettersMajuscules.contains(lettre)) {
              nbMajLeft--;
        } else {
              if (listLettersMinuscules
                          .contains(lettre)) {
                    nbMinLeft--;

              }
        }
        }
                                                                                    }
                                                                                }
                                                                            }
                                                                        }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
              }
        }
        return password;

}
```

Annexe D : Code source de la classe « ExtranetClientUtils »

```java
package com.doandgo.extranetclient.util;

/**
 * Classe utilitaire qui comporte toutes les variables liées au projet
 * @author Thomas CHARMES
 */
public class ExtranetClientUtil {

	public static final String CATALOG_NAME = "User";
	public static final String TABLE_NAME = "ConfigurationLoginEtMdp";
	public static final String TABLE_CONFIGURATION_VALUE_FIELD_SYS_TITLE =
"ConfigurationLoginMdp";
	public static final String ORGANIZATION_NAME = "DefaultOrganization";
	public static final String APPLICATION_NAME = "ExtranetClient";
	public static final String MOOVAPPS_EXTERNAL_CONNEXION = "MoovappsDS";

	// TABLE CONFIGURATION FIELDS
	public static final String TABLE_CONFIGURATION_FIELD_VALUE_SOCIETE_TYPE_LOGIN =
"societe";
	public static final String TABLE_CONFIGURATION_FIELD_TYPE_LOGIN = "TypeDeLogin";
	public static final String TABLE_CONFIGURATION_FIELD_TYPE_PASSWORD = "TypeMotDePasse";
	public static final String TABLE_CONFIGURATION_FIELD_VALUE_EMAIL_TYPE_LOGIN = "email";
	public static final String TABLE_CONFIGURATION_FIELD_VALUE_NOM_TYPE_LOGIN =
"raisonsociale";
	public static final String TABLE_CONFIGURATION_FIELD_NB_TOTAL_CHARS = "nbCharsTotal";
	public static final String TABLE_CONFIGURATION_FIELD_NB_MAJUSCULES =
"NombreExactDeMajuscules";
	public static final String TABLE_CONFIGURATION_FIELD_NB_MINUSCULES =
"NombreExactDeMinuscules";
	public static final String TABLE_CONFIGURATION_FIELD_NB_SPECIAL_CHARS= "nbSpecialChars";
	public static final String TABLE_CONFIGURATION_FIELD_NB_NUMBERS =
"NombreExactDeChiffres";
	public static final String TABLE_CONFIGURATION_FIELD_PREFIXE = "prefixe";
	public static final String TABLE_CONFIGURATION_FIELD_TYPE_PASSWORD_VALUE_STRONG =
"securise";
	public static final String TABLE_CONFIGURATION_FIELD_TYPE_PASSWORD_VALUE_WEAK = "simple";
	public static final String TABLE_CONFIGURATION_FIELD_NB_CHIFFRES_SUFFIXE =
"nbChiffresSuffixe";
	public static final String TABLE_CONFIGURATION_FIELD_INDEX_DEPART = "indexDepart";

	// FORMULAIRE CREATION UTILISATEUR
	public static final String TABLE_FIELD_LOGIN = "Identifiant";
	public static final String TABLE_FIELD_MOT_DE_PASSE = "MotDePasse";
	public static final String TABLE_FIELD_CONFIRMATION_MOT_DE_PASSE =
"ConfirmationMotDePasse";
	public static final String TABLE_FIELD_NOM = "Nom";
	public static final String TABLE_FIELD_PRENOM = "Prenom";
	public static final String TABLE_FIELD_EMAIL = "Email";
	public static final String TABLE_FIELD_METHOD_GESTION_ID_PWD =
"MethodeDeGestionDeLIdentifiantEtMotDePasse";
	public static final String TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_MANUEL = "Manuel";
	public static final String TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_SEMI_AUTOMATIQUE =
"semi automatique";
	public static final String TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_AUTOMATIQUE =
"automatique";
	public static final String TABLE_BUTTON_CREER_UTILISATEUR = "CreerLUtilisateur";
	public static final String TABLE_FIELD_ADRESSE_SIEGE_SOCIAL = "AdresseDuSiegeSocial";
	public static final String TABLE_FIELD_ADRESSE_FACTURATION = "AdresseDeFacturation";
	public static final String TABLE_FIELD_ADRESSE_AUTRES_SITES = "AdressesDesAutresSites";

	// ADMIN CREATION CLIENT
	public static final String ORGANIZATION_TYPE_EXTERNAL = "External";
	public static final String CLIENTS = "Customers";
	public static final String DISTRIBUTEURS = "Distributors";
	public static final String FOURNISSEURS = "Suppliers";
	public static final String SOUS_TRAITANTS = "SubContractors";
```

```
	// EXTENDED ATTRIBUTES CLIENT
	public static final String CLIENT_EXTENDED_ATTRIBUTE_ADRESSE_FACTURATION =
"BillingAddress";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_ADRESSE_SIEGE_SOCIAL =
"HeadOfficeAddress";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_ADRESSE_AUTRES_SITES =
"OtherSitesAddress";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_CODE_NAF =
"ActivityClassificationSystemCode";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_CONTACT_INTERNE =
"ContactsOfTheProducer";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_CONTACT_PRINCIPAL =
"MainContactOfTheCustomer";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_LOGO = "Logo";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_NUMERO_DUNS = "DunsNumber";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_NUMERO_CLIENT = "CustomerNumber";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_NUMERO_SIRET =
"CompanyRegistrationNumber";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_PERSONNALISATION = "Branding";
	public static final String CLIENT_EXTENDED_ATTRIBUTE_QUOTAS_UTILISATEURS =
"QuotasUtilisateurs";

	// FORMULAIRE CREATION CLIENT
	public static final String FIELD_CATEGORIE_CLIENT = "CategorieDeClient"; // Organization
de destination
	public static final String FIELD_CLIENT = "Client"; // Champ de retour du client
fraîchement créé de type organization
	public static final String FIELD_VALIDATION_CLIENT_NECESSAIRE =
"ValidationDuClientNecessaire";
	public static final String FIELD_NUMERO_CLIENT = "NumeroClient";
	public static final String FIELD_QUOTA_UTILISATEURS = "NombreDUtilisateursPrevus";
	public static final String FIELD_CODE_NAF = "CodeNAF";
	public static final String FIELD_NUMERO_DUNS = "NoDUNS";
	public static final String FIELD_NUMERO_SIRET = "NSIRET";
	public static final String FIELD_COLLABORATEURS_EN_CHARGE_DU_DOSSIER =
"CollaborateursEnChargesDuDossier";

}
```

Annexe E : Code source de la classe « CheckConfiguration »

```java
package com.doandgo.extranetclient.user;

import com.axemble.vdoc.sdk.document.extensions.BaseStorageResourceExtension;
import com.axemble.vdoc.sdk.interfaces.IResourceController;
import com.axemble.vdoc.sdk.interfaces.IStorageResource;
import com.doandgo.extranetclient.util.ExtranetClientUtil;
import com.doandgo.extranetclient.util.InvalidArgumentValueException;
import com.doandgo.extranetclient.util.UserUtils;
import com.doandgo.moovapps.utils.NullUtils;

/**
 * Classe qui vérifie la cohérence des données saisies dans la table configuration :
 * L'index de départ ne doit pas excéder 999 999
 * Le nombre maximal de chiffres autorisé pour le suffixe est de 6
 *
 * @author Thomas CHARMES
 *
 */
public class CheckConfiguration extends BaseStorageResourceExtension {

   private static final long serialVersionUID = 1L;

   @Override
   public boolean onBeforeSave() {

          IStorageResource storageResource = getStorageResource();
          IResourceController controller = getResourceController();

          String typeLogin = (String) storageResource.getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_TYPE_LOGIN
          String typeValueSociete = ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_VALUE_SOCIETE_TYPE_LOGIN;

          if (typeLogin.equals(typeValueSociete)) {
                 Float indexDepart = (Float) storageResource
                               .getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_INDEX_DEPART);
                 Float nbChiffresSuffixe = (Float) storageResource
                               .getValue(ExtranetClientUtil.TABLE_CONFIGURATION_FIELD_NB_CHIFFRES_SUFFIXE);

                 if (NullUtils.isNull(indexDepart)) {
                        indexDepart = 1F;
                 }

                 if (indexDepart > 999999) {
                        controller.alert(
                                      getWorkflowModule().getStaticString("LG_TRAD_STARTED_INDEX_TOO_HIGH"));
                        return false;
                 }

                 if (NullUtils.isNull(nbChiffresSuffixe)) {
                        nbChiffresSuffixe = 0F;
                 }

                 if (nbChiffresSuffixe > 6) {
                        controller.alert(
                                      getWorkflowModule().getStaticString("LG_TRAD_SUFFIXE_TOO_LONG"));
                        return false;
                 }

                 boolean donneesValides = false;

                 try {
                        donneesValides = UserUtils.indexContientMoinsDeChiffresQueLeSuffixe(indexDepart.intValue(),
                                      nbChiffresSuffixe.intValue());
                 } catch (InvalidArgumentValueException e) {
                        e.printStackTrace();
                        return false;
                 }
                 if (!donneesValides) {
                        controller.alert(
                                      getWorkflowModule().getStaticString("LG_TRAD_INVALID_DATA"));
                 }
                 return donneesValides;
          }
          return super.onBeforeSave();
```

Annexe F : Code source de la classe « CheckPasswordAllowed »

```java
package com.doandgo.extranetclient.user;

import com.axemble.vdoc.sdk.document.extensions.BaseDocumentExtension;
import com.axemble.vdoc.sdk.interfaces.IAction;
import com.axemble.vdoc.sdk.interfaces.IResourceController;
import com.axemble.vdoc.sdk.interfaces.IWorkflowInstance;
import com.doandgo.commons.utils.RegexpUtils;
import com.doandgo.extranetclient.util.ExtranetClientUtil;
import com.doandgo.moovapps.utils.NullUtils;

/**
 * Classe qui vérifie que le mot de passe comporte bien 8 caractères et commence par une majuscule.
 * Positionnée sur le sous-formulaire d'étape "Création de l'utilisateur" et appelée avant l'enregistrement
 * ou le passage à l'étape suivante.
 * @author Thomas CHARMES
 */
public class CheckPasswordAllowed extends BaseDocumentExtension {
	private static final long serialVersionUID = 1L;

	@Override
	public boolean onBeforeSave() {
		return execute();
	}

	@Override
	public boolean onBeforeSubmit(IAction action) {
		if (action.getName().equals(ExtranetClientUtil.TABLE_BUTTON_CREER_UTILISATEUR)) {
			return execute();
		}
		return super.onBeforeSubmit(action);
	}

	private boolean execute() {
		IWorkflowInstance wi = getWorkflowInstance();
		if (NullUtils.isNull(wi)) {
			reportError("wi null");
			return false;
		}
		IResourceController controller = getResourceController();
		return checkPasswordIsAllowed(wi, controller);
	}

	private boolean checkPasswordIsAllowed(IWorkflowInstance wi, IResourceController controller) {
		String methodeGestionLoginEtMdp = (String)
wi.getValue(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD);
		String motDePasse = (String) wi.getValue(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE);
		if
(!methodeGestionLoginEtMdp.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_AUTOMATIQUE)) {
			if (NullUtils.isNull(motDePasse)) {
				reportError("Mot de passe null");
				return false;
			}
			if (!startByUpperCaseAndHasMinimumEightCharacters(motDePasse)) {

	controller.alert(getWorkflowModule().getStaticString("LG_TRAD_UNAUTHORIZED_PASSWORD"));
				return false;
			}
		}
		return true;
	}

	private boolean startByUpperCaseAndHasMinimumEightCharacters(String motDePasse) {
		if (!RegexpUtils.isStartByUppercase(motDePasse))
			return false;
		// Le mot de passe doit avoir au moins 8 caractères
		if (RegexpUtils.isMoreThanNCharacters(motDePasse, 7))
			return false;
		return true;
	}

	private static void reportError(String log) {
		LOGGER.info(log);
		System.out.println(log);
	}
}
```

Annexe G : Code source de la classe « CheckPasswordMatchesWithConfirmation »

```java
package com.doandgo.extranetclient.user;

import com.axemble.vdoc.sdk.document.extensions.BaseDocumentExtension;
import com.axemble.vdoc.sdk.interfaces.IAction;
import com.axemble.vdoc.sdk.interfaces.IResourceController;
import com.axemble.vdoc.sdk.interfaces.IWorkflowInstance;
import com.doandgo.extranetclient.util.ExtranetClientUtil;
import com.doandgo.moovapps.utils.NullUtils;

/**
 * Classe qui vérifie que le mot de passe tapé est bien identique à la
 * confirmation. Positionnée sur le sous-formulaire d'étape "Création de
 * l'utilisateur" et appelée avant l'enregistrement ou le passage à l'étape
 * suivante.
 *
 * @author Thomas CHARMES
 */
public class CheckPasswordMatchesWithConfirmation extends BaseDocumentExtension {
	private static final long serialVersionUID = 1L;

	@Override
	public boolean onBeforeSave() {
		return execute();
	}

	@Override
	public boolean onBeforeSubmit(IAction action) {
		if (action.getName().equals(ExtranetClientUtil.TABLE_BUTTON_CREER_UTILISATEUR)) {
			return execute();
		}
		return super.onBeforeSubmit(action);
	}

	private boolean execute() {
		IWorkflowInstance wi = getWorkflowInstance();
		if (NullUtils.isNull(wi)) {
			reportError("wi null");
			return false;
		}
		IResourceController controller = getResourceController();
		return passwordMatchesWithConfirmation(wi, controller);
	}

	private boolean passwordMatchesWithConfirmation(IWorkflowInstance wi, IResourceController
controller) {
		String methodeGestionLoginEtMdp = (String)
wi.getValue(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD);
		if
(methodeGestionLoginEtMdp.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_AUTOMATIQUE))
			return true;
		String motDePasse = (String) wi.getValue(ExtranetClientUtil.TABLE_FIELD_MOT_DE_PASSE);
		String confirmation = (String)
wi.getValue(ExtranetClientUtil.TABLE_FIELD_CONFIRMATION_MOT_DE_PASSE);
		if (NullUtils.isAtLeastOneNull(motDePasse, confirmation)) {
			reportError("mdp et/ou confirmation null(s)");
			controller.alert(
					getWorkflowModule().getStaticString("LG_TRAD_EMPTY_PASSWORD"));
			return false;
		}
		if (!motDePasse.equals(confirmation)) {
			controller.alert(
					getWorkflowModule().getStaticString("LG_TRAD_DIFFERENT_CONFIRMATION"));
			return false;
		}
		return true;
	}

	private static void reportError(String log) {
		LOGGER.info(log);
		System.out.println(log);
	}
}
```

Annexe H : Code source de la classe « CheckUniqueLogin »

```
package com.doandgo.extranetclient.user;

import com.axemble.vdoc.sdk.document.extensions.BaseDocumentExtension;
import com.axemble.vdoc.sdk.interfaces.IAction;
import com.axemble.vdoc.sdk.interfaces.IResourceController;
import com.axemble.vdoc.sdk.interfaces.IWorkflowInstance;
import com.doandgo.extranetclient.util.ExtranetClientUtil;
import com.doandgo.extranetclient.util.UserUtils;
import com.doandgo.moovapps.utils.NullUtils;

public class CheckUniqueLogin extends BaseDocumentExtension {
	private static final long serialVersionUID = 1L;

	@Override
	public boolean onBeforeSave() {
		return execute();
	}

	@Override
	public boolean onBeforeSubmit(IAction action) {
		if (action.getName().equals(ExtranetClientUtil.TABLE_BUTTON_CREER_UTILISATEUR)) {
			return execute();
		}
		return super.onBeforeSubmit(action);
	}

	private boolean execute() {
		IWorkflowInstance wi = getWorkflowInstance();
		if (NullUtils.isNull(wi)) {
			reportError("wi null");
			return false;
		}
		IResourceController controller = getResourceController();
		return loginIsUnique(wi, controller);
	}

	private boolean loginIsUnique(IWorkflowInstance wi, IResourceController controller) {
		boolean isUnique = true;
		String methodeGestionLoginEtMdp = (String)
wi.getValue(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD);
		String login = (String) wi.getValue(ExtranetClientUtil.TABLE_FIELD_LOGIN);
		if (NullUtils.isNull(login)) {
			reportError("login null");
			isUnique = false;
		}
		if
(methodeGestionLoginEtMdp.equals(ExtranetClientUtil.TABLE_FIELD_METHOD_GESTION_ID_PWD_VALUE_MAN
UEL)) {
			if (UserUtils.alreadyExistsInDirectoryModule(login)) {

	controller.alert(getWorkflowModule().getStaticString("LG_TRAD_UNIQUE_LOGIN"));
				isUnique = false;
			}
		}
		return isUnique;
	}

	private static void reportError(String log) {
		LOGGER.info(log);
		System.out.println(log);
	}
}
```

Annexe I : Code source de la classe « CheckQuotas »

```java
package com.doandgo.extranetclient.user;

import java.util.Collection;

import com.axemble.vdoc.sdk.document.extensions.BaseDocumentExtension;
import com.axemble.vdoc.sdk.exceptions.DirectoryModuleException;
import com.axemble.vdoc.sdk.interfaces.IAction;
import com.axemble.vdoc.sdk.interfaces.IContext;
import com.axemble.vdoc.sdk.interfaces.IOrganization;
import com.axemble.vdoc.sdk.interfaces.IResourceController;
import com.axemble.vdoc.sdk.interfaces.IUser;
import com.axemble.vdoc.sdk.interfaces.IWorkflowInstance;
import com.axemble.vdoc.sdk.modules.IDirectoryModule;
import com.doandgo.extranetclient.util.ExtranetClientUtil;
import com.doandgo.moovapps.utils.NullUtils;

public class CheckQuotas extends BaseDocumentExtension {
	private static final long serialVersionUID = 1L;

	@Override
	public boolean onBeforeSubmit(IAction action) {

		if (action.getName().equals(ExtranetClientUtil.TABLE_BUTTON_CREER_UTILISATEUR)) {
			try {
				return execute();
			} catch (DirectoryModuleException e) {
				e.printStackTrace();
			}
		}
		return super.onBeforeSubmit(action);
	}

	private boolean execute() throws DirectoryModuleException {

		IWorkflowInstance wi = getWorkflowInstance();
		if (NullUtils.isNull(wi)) {
			reportError("wi null");
			return false;
		}
		IResourceController controller = getResourceController();
		return quotasIsNotFull(wi, controller);
	}

	private boolean quotasIsNotFull(IWorkflowInstance wi, IResourceController controller)
			throws DirectoryModuleException {

		int nbUsersForThisCustomer = 0;
		IDirectoryModule dm = getDirectoryModule();
		IContext iContext = getWorkflowModule().getSysadminContext();
		IOrganization client = (IOrganization) wi.getValue(ExtranetClientUtil.FIELD_CLIENT);
		Collection<? extends IUser> users = dm.getUsers(iContext, client);
		for (@SuppressWarnings("unused") IUser iUser : users) {
			nbUsersForThisCustomer ++;
		}
		Float quotas = (Float) client.getExtendedAttributes()
				.getValue(ExtranetClientUtil.CLIENT_EXTENDED_ATTRIBUTE_QUOTAS_UTILISATEURS);
		if (! NullUtils.isNullOrEmpty(quotas) && nbUsersForThisCustomer >= quotas.intValue()) {
			getResourceController().alert(getWorkflowModule().getStaticString("LG_TRAD_QUOTAS"));
			return false;
		}
		return true;
	}

	private static void reportError(String log) {
		LOGGER.info(log);
		System.out.println(log);
	}
}
```

Annexe J : Code source du fichier xml d'internationalisation « doandgo-extranetclient.xml »

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- In order to add or overwrite string values, please, -->
<!-- create your own XML strings file (e.i. project-resources.xml)". -->
<res>
	<id value="LG_PROJECT_STRING">
		<lang value="fr" flag="0">Ma chaine spécifique</lang>
		<lang value="en" flag="0">My specific string</lang>
	</id>
	<id value="LG_PROJECT_EXTRANETCLIENT">
		<lang value="fr" flag="0">Extranet client</lang>
		<lang value="en" flag="0">Client Extranet</lang>
	</id>
	<id value="LG_TRAD_INCOHERENT_ARGS">
		<lang value="fr" flag="0">Attention ! Les arguments de sont pas cohérents.</lang>
		<lang value="en" flag="0">Warning ! Arguments are not available.</lang>
	</id>
	<id value="LG_TRAD_QUOTAS">
		<lang value="fr" flag="0">Attention ! Vous ne pouvez plus créer d'utilisateurs pour ce
client. Le quota maximal a été atteint.</lang>
		<lang value="en" flag="0">Warning ! You can't create user for this customer any more. The
maximum number of users has been reached.</lang>
	</id>
	<id value="LG_TRAD_UNIQUE_LOGIN">
		<lang value="fr" flag="0">Attention ! L'identifiant saisi existe déjà. Merci de réitérer la
saisie.</lang>
		<lang value="en" flag="0">Warning ! This login already exists. Please enter an other
one.</lang>
	</id>
	<id value="LG_TRAD_EMPTY_PASSWORD">
		<lang value="fr" flag="0">Attention ! Merci de saisir un mot de passe ainsi que sa
confirmation.</lang>
		<lang value="en" flag="0">Warning ! Please enter a password and its confirmation.</lang>
	</id>
	<id value="LG_TRAD_DIFFERENT_CONFIRMATION">
		<lang value="fr" flag="0">Attention ! Le mot de passe saisi est différent de la
confirmation. Merci de réitérer la saisie.</lang>
		<lang value="en" flag="0">Warning ! The entered password is different from the confirmation.
Please enter it again.</lang>
	</id>
	<id value="LG_TRAD_UNAUTHORIZED_PASSWORD">
		<lang value="fr" flag="0">Attention ! Le mot de passe saisi n'est pas autorisé. Merci de
réitérer la saisie.</lang>
		<lang value="en" flag="0">Warning ! The entered password is not allowed. Please enter it
again.</lang>
	</id>
	<id value="LG_TRAD_STARTED_INDEX_TOO_HIGH">
		<lang value="fr" flag="0">Attention ! L'index de départ ne peut pas excéder 999 999. Merci
de réitérer la saisie.</lang>
		<lang value="en" flag="0">Warning ! The entered started index can't exceed 999 999. please
enter a lower value.</lang>
	</id>
	<id value="LG_TRAD_SUFFIXE_TOO_LONG">
		<lang value="fr" flag="0">Attention ! Le nombre de chiffres du suffixe est trop élevé. Merci
de saisir un chiffre strictement inférieur à 7.</lang>
		<lang value="en" flag="0">Warning ! The login is too long. Please select a number of numbers
for the suffix strictly inferior to 7.</lang>
	</id>
	<id value="LG_TRAD_INVALID_DATA">
		<lang value="fr" flag="0">Attention ! Le suffixe est plus long que le nombre de chiffres de
l'index de départ. Merci de réitérer la saisie.</lang>
		<lang value="en" flag="0">Warning ! The suffix is longer than the starting index's number of
numbers. Please enter new values.</lang>
	</id>
		<id value="LG_TRAD_ADDRESSES">
		<lang value="fr" flag="0">Ajouter les adresses aux attributs étendus</lang>
		<lang value="en" flag="0">Add addresses to extended attributes</lang>
	</id>
</res>
```

RÉSUMÉ EN FRANÇAIS

Ce rapport détaille les différentes phases de l'élaboration d'un projet consistant à réaliser un extranet destiné aux clients. Aujourd'hui, de nombreuses entreprises souhaitent restructurer leur système d'information et/ou leur système informatique afin d'évoluer progressivement vers le « zéro papier ». Cette transition numérique peut passer par la création d'une GED (Gestion Electronique des Documents) ou encore par l'utilisation de processus visant d'une part à informatiser les demandes internes au sein de l'entreprise et d'autre part à accélérer leurs temps de traitement. Pour accéder à ces services, un utilisateur doit obligatoirement avoir un compte. L'objectif principal de ce projet est de faciliter la création et le suivi des comptes utilisateurs par le gestionnaire. Pour cela, une application répondant à ce besoin a été développée. Ce rapport détaille la partie de l'application relative à la génération de l'identifiant et du mot de passe d'un utilisateur lors de la création de ce dernier.

MOTS CLÉS : processus – Moovapps – utilisateur – configuration – identifiant – mot de passe

ENGLISH SUMMARY

This report details the multiple stages of the decision-making process in order to achieve the development of a customer extranet. Today, a lot of companies want to utterly rebuild their information/informatic system in order to move towards « zero paper » system. This digital transition can be represented by the creation of an EDM (Electronic Document Management) or by using several digital processes to manage internal requests within a company and then accelerate their treatment time. To acces those services, users must have an account. The main goal of this project is to facilitate the creation and managing of customers and users by the manager. For that, an application answering those needs has been developped. This report details the part of this application dedicated to the generation of the login and the password for a user during the creation process of it.

KEY WORDS : process – Moovapps – user – configuration – login – password