Analyse Technique

Création d'un logiciel de gestion d'une société

Contexte

Le Projet traité aborde la création d'un programme permettant de gérer une société via différents outils simples. Ce programme est destiné à un nombre restreints d'utilisateurs et est voué à être utilisé de manière locale ou via un Cloud.

En outre, il doit être capable d'assigner aux différents utilisateurs des rôles distincts leurs permettant d'accéder à des fonctions propres, tout en s'assurant que les autres utilisateurs ne puissent pas gérer des paramètres qui ne les concernent pas.

Il en découle en certains nombres d'enjeux à résoudre pour accomplir ce projet, notamment la sécurité et la gestion des droits, la persistance des données ainsi que la mise en place de l'interface utilisateur.

Persistance des données

Bien qu'il soit tout à fait possible d'imaginer un programme fonctionnant uniquement sur la base de fichiers à charger / sauvegarder via input/output - et donc que chaque nouvelle connexion engendrerait un programme vierge - cela serait fastidieux de maintenir un suivi global de la société cible. Une altération de ces fichiers pourrait également causer une perte définitive des données utilisées.

Pour une utilisation plus efficace, il sera nécessaire que le programme puisse charger automatiquement l'état des valeurs entrées lors des dernières utilisations. L'utilisateur sera à même de mettre directement à jour celles-ci ou de charger des résultats différents. Pour se faire, il existe différentes méthodes.

Système de fichiers et XML

Il est tout à fait envisageable que le programme puisse enregistrer automatiquement les données de son utilisation actuelle dans des fichiers dédiés, puis de les lire automatiquement à chaque nouvelle ouverture du programme. L'extension et le format de ces fichier importe peu du moment que le programme soit cohérant entre l'écriture et la lecture des données.

Il existe quelques méthodes implémentées dans les librairies JAVA permettant de faciliter cette tâche. On notera en particulier les méthodes liées aux préférences (**Preference API**) qui permettent de stocker facilement des paires de données simples vers un dossier spécifié.

Cette méthode possède un certain nombre de faiblesses :

- Les fichiers se doivent d'être sécurisés et ne sont pas à l'abri d'altération externes qui pourraient les invalider complétement ou insérer des données inconsistantes.
- Ces données sont souvent stockées en « langage humain », ce qui permet à n'importe qui d'y accéder sans passer par l'application.
- La lecture peut se montrer extrêmement fastidieuse à mettre en place suivant la complexité des éléments et des objets à initialiser. Les données sont principalement déstructurées dans ce type de fichier.

Ce dernier point peut cependant être corrigé en utilisant des librairies capables de « décomposer » les objets en des ensembles de propriétés. Ces fichiers XML sont plus structurées et plus maniables que des fichiers d'autres extensions, la transition objet -> XML , XML-> objet est d'ailleurs facilité par des librairies telle que **XStream**, par exemple. Cependant, quand les données deviennent encore plus complexes, où que des traitements préalables sont nécessaires (par exemple, fournir une liste de mouvements bancaires en les filtrant par période), le système de fichier demande l'implémentation de méthodes supplémentaires redondantes.

Bases de données

Il est également possible d'opter pour les bases de données quand il s'agit de stocker des données de manière plus sûre. On en avancera deux types : les bases de données placées sur un serveur ainsi que les bases de données intégrées.

Le premier type a l'avantage d'être accessible par tous via différentes copies de l'application, et de fournir un stockage prenant en compte toutes les modifications simultanées : il permet de gérer plusieurs utilisateurs en même temps. Cependant, il est peu adapté à la problématique en cours : le programme sera utilisé localement par un nombre restreint d'utilisateurs – un à la fois - et l'accès au serveur n'est pas forcément assuré lors de son utilisation.

La base de données intégrée dans l'application perd l'avantage du traitement des flux simultanées et du stockage externe, mais gagne celui de la flexibilité : L'application traite elle-même ses données, et aucun autre applications n'est nécessaire pour la faire fonctionner.

Il existe un certain nombre de database entièrement conçue en Java, mais on notera essentiellement **H2**, **Apache Derby** et **HSQLDB**. Ces trois moteurs permettent d'intégrer les deux types de bases de données citées. Le programme pour ce projet ne nécessitant pas de stocker un nombre incommensurable de donnée, les différences d'intégration restent plutôt minimes. **Derby** est réputé plus simple d'utilisation mais plus lent, **HSQLDB** est adapté à la

suite OpenOffice mais semble plus instable tandis que **H2** offre un large éventail de possibilités mais en devient donc un petit peu plus compliqué à prendre en main que ses congénères.

Ces trois bases de données possèdent une bonne documentation, mais seul Apache Derby a été essayé durant cette analyse.

Cette méthode ne se protège pas des altérations / suppressions des données, mais permet de décomposer des objets complexes en des éléments simples et de les reconstituer facilement. En outre, il offre une première sécurité : les données sont stockées en langage « machine ». Stocker des données cryptées est tout à fait envisageable. Le plus grand avantage vis-à-vis des autres technologies reste la possibilité de manipuler facilement les données, directement depuis la base de données.

Comme le programme nécessite la gestion de liste, de tableau, et de filtrage de donnée, cette méthode semble donc la plus adaptée.

Sécurité et Gestion des droits

Ce programme nécessite quelques points incontournables :

- Un système d'authentification pour autoriser seulement les membres du comité à avoir accès aux données de l'application
- Un système de rôle fournissant des permissions aux divers utilisateurs
- Un « super administrateur » ou un fichier de gestion capable d'ajouter des utilisateurs et modifier leurs droits

Encore une fois il existe plusieurs alternatives :

Fichiers de gestion

Il est possible de gérer ces points à l'aide de fichiers simples et de variables dans le code source. Cela revient à utiliser un système de lecture et d'écriture uniquement.

Cependant, une gestion de ce type créerait des tables de permissions « implicites » directement dans le code, ce qui pourrait poser des problèmes si des changements doivent être effectués dans le code source. Ce système peut être utilisable pour gérer des *rôles* mais est peu adapté pour gérer les *permissions*. L'idéal serait de modéliser des classes définissants les utilisateurs : ce travail qui peut se montrer fastidieux existe déjà sous la forme de Framework ou de libraire. De plus des fichiers de gestions laissent apparaître des mots de passes, des utilisateurs et des rôles en « clair ». Bien que le programme soit utilisé dans des espaces de confiance, cela reste le niveau 0 de la sécurité.

Base de données

Il est possible de sécuriser ces informations en cryptant les données et en les plaçant dans une base de données sécurisée et persistante. Bien que optimal, cela poserait toujours des problèmes quand il s'agira d'effectuer des modifications sur les permissions, la sécurité se faisant à nouveau de manière « implicite ». L'usage de base de données ne permet pas non plus de définir efficacement l'utilisateur.

Cet outil peut servir cependant à la résolution de plusieurs problèmes, comme il a été montré pour le stockage des données et sera probablement utilisé dans ce projet.

Framework

Il existe des framework de sécurité permettant de gérer efficacement et simplement l'authentification et la gestion de droits tout en permettant de définir l'objet « utilisateur ». Pour cette analyse, **Apache Shiro** a été utilisé. Bien qu'il en existe d'autres plus complets, Shiro reste l'un des outils du genre le plus facile à manipuler. Il offre entre autre la possibilité de définir efficacement les rôles et les permissions de manières explicites, où d'ajouter des annotations directement sur les méthodes concernées par des autorisations.

Son défaut revient au fait que les mots de passes, utilisateurs et les rôles sont stockés dans un fichier externe qui devra être crypté ou protégé par un mot de passe.

La solution optimale serait alors de mettre en place une framework s'occupant de la gestion des droits et de stocker les données sensibles dans une base de données sécurisée pour profiter des avantages de chacune des méthodes.

Compte administrateur

Il faut aussi signaler que si le programme possède un système d'authentification, il faut que les utilisateurs aient des « comptes ». Le programme étant prévu à un usage définit de personne et que les nouvelles « inscriptions » doivent être prévues par le comité usant le logiciel, un compte administrateur sera nécessaire. Il pourra alors créer à l'avance des utilisateurs en ayant accès à des panels qui lui seront réservés, et assigner des rôles – par extension des permissions – à chacun des membres du comité.

Interface graphique

L'application demande une interface simple composée de boutons et de champs de textes. Il n'est donc pas nécessaire d'avoir un outil graphique trop avancé.

Il existe de très nombreuses alternatives.

Swing

Swing est l'outil graphique le plus adapté à ce type de problème. Il offre un riche éventail d'objets et de dispositions de « layout » permettant d'effectuer facilement la grande majorité des interfaces simples.

Cet outil a été maitrisé.

Apache Pivot

Pivot permet de créer des interfaces riches et dynamiques. Il contient de très nombreux outils mais est conçu et plus adapté pour les applications basées sur le web.

JavaFX

Outil ayant supplanté Swing, il propose des outils plus avancés en étant capable de gérer vidéo, audio et 3D. Il offre de très nombreuses possibilités.

Chacune de ces options peuvent être étendues à l'aide de libraires spécialisées, qui ne seront probablement pas nécessaire dans l'optique de cette application.

Il en existe bien d'autres mais elles seront équivalentes sur une interface simple de ce type. L'application se concentre sur l'utilité de ses outils plutôt que celui de son design. Il est plus judicieux d'afficher un outil à la fois pour concentrer l'information utile plutôt que de noyer l'écran de l'ensemble des résultats.