FRAMEWORK LADO

KWATE DASSI Loïc

Contents

0.1 Introduction

Le développement d'un bon logiciel est un exercice qui demande beaucoup à la ressource humaine qui a la charge du projet. les développeurs de logiciel sont presque tout le temps victimes de la flexibilité des exigences des clients qui parfois ne savent même pas ce qu'ils veulent. Résoudre les problèmes liés à la flexibilité aléatoire des exigences des clients nous amène à développer un framework orienté objet **LADO** qui a pour but de faire du développement générique du logiciel, un développement qui est beaucoup plus axé sur la paramétrisation du noyau que de développement de celui. Ce framework nous permettra de faire un développement plus fiable, robuste, flexible, rapide et sécurisé de nos logiciels.

0.2 Spécifications

0.2.1 Robustesse

0.2.2 Fiabilité

0.2.3 Fiabilité

0.2.4 Sécurité

0.3 Fonctionnement

0.3.1 Base de données

Dans le développement du logiciel la chose la plus important est **la donnée (la forme de sauvegarde), l'accessibilité, la sécurité,....** Le premier point sur lequel nous allons nous appuyer est la donnée car elle constitue la base principale de notre noyau. La base de donnée est le principal acteur dans le fonctionnement du framework, elle regorge toutes les informations dont le framework aura besoin pour générer de façon générique les logiciels.

Structure basique de la base de données

les informations qui sont communes à toutes les applications seront regroupées dans les tables suivantes: **utilisateur, groupe, operation, droit, type-operation, historiquecreateelement, localisation, tablecode, sexe, typeutilisateur, clechiffrement, statut, historiqueupdateelement, historiquesuppressionelement**

0.3.2 Algorithme

cette section regroupe les algorithmes (et sommairement les fonctionnement) principaux sur lesquels le framework est bâti

LADO-ALGO-1000 : Chiffrement

principe de fonctionnement :
nous partons sur la base que le logiciel doit garanti une sécurité infaillible à priori, et nous allons nous baser sur les algorithmes de chiffrement des messages entre la base de données et l'application

LADO-ALGO-1001 : Calcul de droit

Tout système a des utilisateurs, ceux-ci sont susceptibles d'être dans des groupes. Un système sécurisé voudrait donc qu'en son sein règne une logique d'assignation de droit. Les principales règles d'assignation des droits aux utilisateurs sont les suivantes:

- les droits ne sont pas directement affectés aux utilisateurs mais aux groupes auxquels ils appartiennent
- un groupe n'a pas une droit qui ne lui a pas été attribué
- les droits d'un utilisateur sont calculés en faisant réunions de tous les droits des groupes auxquels cet utilisateur appartient
- un droit est attribué sur une opération

soit O_1 une opération dans le système, g un groupe crée au sein du système, $isGroupAllow(g, O_1)$ la fonction qui prend en paramètre un groupe et un droit et renvoie $True$ si le groupe g a le droit et la $False$ si non, soient u un utilisateur donné du système, $isUserAllow(u, O_1, setGroup)$ la fonction qui prend en paramètre un utilisateur ,un droit, l'ensemble des groupes auxquels l'utilisateur appartient et rend compte l'assignation du droit à l'utilisateur, cette fonction est calculer comme suit :
$isGroupAllow(g_1, O_1)$ *ou* $isGroupAllow(g_2, O_2)$ *ou ... ou* $isGoupAllow(g_n, O_n)$

LADO-ALGO-1002 : chargement dynamique des objets à partir des tables

la structure classique des objets en java n'offre pas une grande flexibilité dans la manipulation de ceux-ci. la structure de donnée utilisé sera similaire au tableur d'excel. l'interface pour cette structure de donnée est la suivante:

- **loadColumns(Collection<? extends Column>)** //chargement des colonnes
- **getMetadata()** // retourne les informations sur la structure de donnée et non sur les données qu'elle contient
- **addRow()** // ajouter une nouvelle ligne au tableur
- **deleteColumn(String)**
- **addColumn(Column)**
- **applyOnColumn(Interface, columnName)** // appliquer un traitement à toute la colonne
- **sortByColumns(columnName, orderwise, comparateur)** // trier les lignes
- **searchRow(columnName, value)** // renvoi la ligne correspondant au critère de recherche
- **DisplayRow(Interface critere, columnname, displayOption : boolean)** // affiche ou cache (en fonction de la valeur de displayOption) les lignes qui vérifient le critère passé en paramètre.
- **DiplayRow(HashMap<String:columnName, Interface critere>)** //le traitement précédent sera appliqué sur plusieurs colonnes
- **DisplayColumns(ColumnName, diplayOption)** // affichage de colonne
- **undoDisplay()** // retourne le tableur dans sa structure initiale
- **aggregateFunction(functionName, columnName)** // functionName le nom d'une fonction d'agrégation (MAX, MIN, SUM, AVG, COUNT, COUNTDISTINCT, STDEV, VAR)