Logique métier

# Introduction

Pour la partie du code concernant la logique métier, nous avons décidé de partir sur une séparation entre les modèles et la vraie logique. Les classes modèles sont principalement utilisées pour le mappage avec la DAL, alors que les classes logiques implémentes vraiment les méthodes qui vont permettre les interactions entre les classes elles-mêmes, les classes des interfaces graphiques et celles de la couche d’accès aux données. Pour faire cette séparation, nous avons défini deux packages, le package **model** (bll.model) et le package **logic** (bll.logic).

Les classes modèles déclarent principalement les attributs en lien avec la base de données, ceux qui doivent donc être mappés et implémentent les méthodes de modifications (setters) et de récupérations (getters) de ces derniers ainsi que les deux méthodes qui permettent d’ajouter et de mettre à jour les données dans les bases de données.

Les classes logiques, quant à elles, implémentent la vraie logique métier de l’application, c’est-à-dire les interactions entre elles, les attributs qui les concernent et les méthodes nécessaires à la partie graphique. Les méthodes logiques contiennent aussi la méthode de suppression des éléments dans les bases de données, qui s’assure de supprimer l’objet dans la base ainsi que tous les liens éventuels contenus dans les autres objets.

# Classes *Authentication* et *KeyGenerator*

La classe *Authentication* est une classe se trouvant dans le package **logic**. Cette dernière n’a pas de modèle car elle ne s’occupe que de la logique concernant l’enregistrement et la connexion d’un utilisateur. Ses tâches principales sont de vérifier la cohérence des informations fournies avec la base de données lors d’un enregistrement ou d’une connexion, de faire le hash du mot de passe et de vérifier la clé envoyée par email lors de la connexion d’un utilisateur non vérifié avec celle enregistrée en base de données, permettant ainsi de valider le compte et l’adresse email fournie.

La classe *KeyGenerator*, de la même manière que la classe *Authentication*, ne possède pas de modèle. Elle permet simplement la génération d’une clé aléatoire, que nous envoyons à l’utilisateur lorsqu’il s’enregistre sur la base de données en ligne.

# Classe *ClientLogic*

ClientLogic est la classe « maîtresse », c’est celle qui conserve les données de l’utilisateur connecté et qui nous permet de retrouver tous les liens. Comme il ne peut y avoir qu’un seul client connecté à la fois, nous avons décidé d’implémenter cette classe d’après le modèle du *Singleton*. Nous ne pouvons donc avoir qu’une seule instance de cette classe lors de l’exécution de notre programme. C’est donc lors de la connexion, si celle-ci passe, que nous paramétrons l’instance et que nous récupérons toutes les données liées à l’utilisateur.

# Les autres classes logiques

En ce qui concerne les autres classes du package **logic**, elles partagent toutes la même structure. Le but principal de ces classes est de bien gérer les interactions entre les différents objets. Certaines de ces classes, comme *CategoryLogic*, sont moins complexes car elles ont moins de liens à gérer. Par exemple dans *CategoryLogic*, nous devons simplement gérer les liens lors de la suppression pour éviter des problèmes de dépendance dans la base de données. Alors que d’autres classes, comme *IOTransactionLogic* ou encore *BudgetLogic*, sont beaucoup plus complexes. Par exemple, pour le cas de *IOTransactionLogic*, lors de la création d’une instance, il fallait mettre à jour tableaux et objets utilisés par la GUI. Lors de la mise à jour d’une instance de cette classe, il fallait vérifier derechef ces tableaux, mettre à jour la somme du compte en banque concerné et ainsi de suite.

Nous pouvons dire que les plus grosses classes au niveau de la complexité sont effectivement *IOTransactionLogic*, du fait de ses nombreux liens à différentes structures, *BudgetLogic*, car nous devions gérer les budgets normaux et les budgets partagés, avec des liens à plusieurs catégories et, dans le cas du deuxième, à plusieurs utilisateurs, ainsi que *DebtLogic*, qui fait un lien entre deux utilisateurs et, lorsqu’une dette est confirmée, nous devions gérer la création d’un transaction pour chacun des utilisateurs concernés.