17/12/2021

Corentin BARAULT

COMPTE RENDU

Gestion de compte bancaire

Table des matières

[Analyse du sujet 2](#_Toc89098529)

[Spécifications des classes 4](#_Toc89098530)

[Evaluation des performances 5](#_Toc89098531)

[Comment compiler le code 6](#_Toc89098532)

[Jeu de test 7](#_Toc89098533)

# Analyse du sujet

*On chercher à développer un système de gestion de comptes bancaires à l’aide d’Akka. Un banquier est en charge de plusieurs comptes, et travaille dans une banque. Lorsqu’un client se présente à la banque pour déposer ou retirer de l’argent, la banque doit transmettre la demande du client au banquier responsable de son compte.*

*Un mécanisme de persistance doit être implémenté en liaison avec un SGBD, afin de pouvoir retrouver l’état des comptes des clients même en cas d’arrêt du système.*

Ce projet dois se réaliser à l’aide de la bibliothèque logiciel Akka qui permet de simplifier la création des application concurrentes. Akka supporte beaucoup de modèles de programmation différents mais est surtout utiliser pour la réalisation du modèle acteur. Akka est un langage écrit en Scala, mais il est utilisable avec Java qui est donc le langage que je vais utiliser pour la réalisation de ce projet. En plus de Java j’utiliserai aussi Maven afin de pouvoir facilement gérer les différentes bibliothèques logiciel du projet.

A partir du sujet, nous pouvons déjà apercevoir les différents acteurs qui appartiendront à au programme.

* Les clients qui auront pour objectifs d’envoyer des messages d’ajout et de retrait à la banque.
* La banque qui aura pour charge de transmettre les messages des clients au banquier ainsi que d’effectuer les différentes transactions sur les comptes.
* Les banquiers qui récupéreront les messages du client qui lui ont été transmis par la banque afin dès les approuver et qui demandera ensuite à la banque d’effectuer la transaction.
* Pour finir, des acteurs de persistances qui eux auront pour but d’enregistrer dans la base de données les nouvelles valeurs des comptes afin d’être sûr que l’on puisse récupérer l’état des comptes mêmes après un arrêt systèmes imprévu.

Voici un schéma montrant comment communique les acteurs entre eux lors d’une demande d’un client.

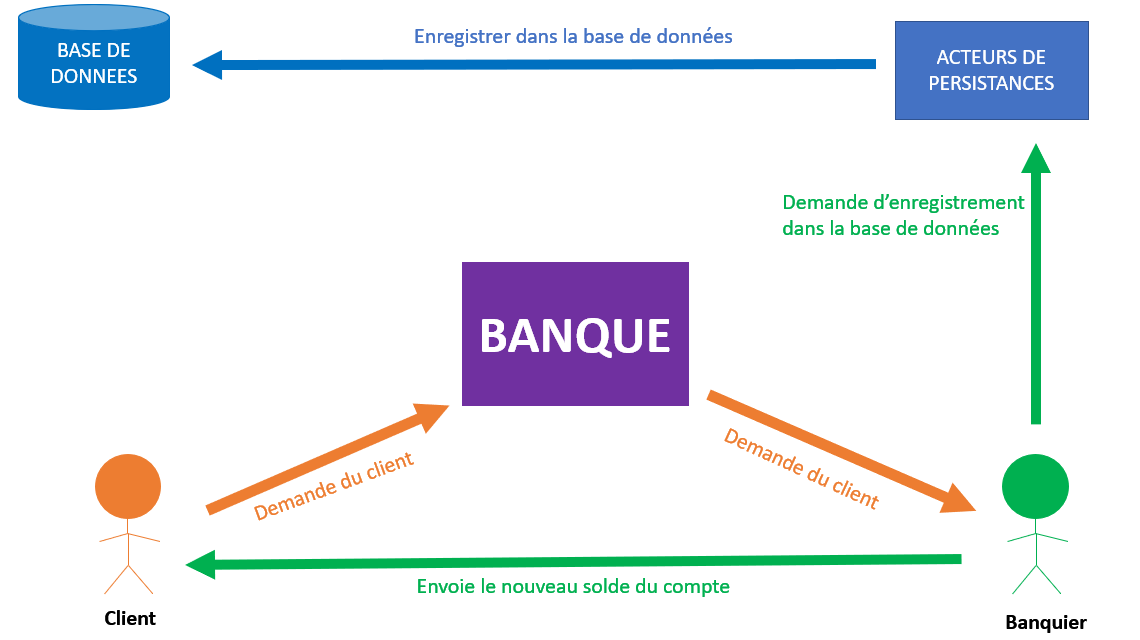


Figure 1: Schéma de communication entre les acteurs lors d’une demande

Cependant certains sous problèmes apparaissent.

***-Comment faire si un banquier reçoit énormément de demande d’un client et qu’effectuer l’ensemble des transactions prennent beaucoup de temps ?***

Pour éviter cela, il faut optimiser le plus possible le code pour que la consommation du message soit assez rapide.

C’est aussi pour cela qu’il est important d’avoir un groupe d’acteur s’occupant uniquement de la persistance dans la base de données, car si un banquier reçoit énormément de transactions et qu’il devait aussi s’occuper d’enregistrer dans la base de données chaque modification, cela ralentirait énormément le programmes cas les enregistrements en base de données sont couteux. Il faut donc un groupe d’acteur dédié qui pourront se partager la charge du travail bet ainsi accéléré le traitement des demandes.

# Spécifications des classes

***Classe Compte :***

Cette classe correspondra à un compte et sera l’élément que les différents acteurs manipuleront.

Elle possédera trois attributs :

* Un attribut id pour que l’on puisse savoir quel compte modifier lors de l’enregistrement des modifications dans la bdd.
* La somme qui sera récupérable à l’aide de la méthode getSomme().
* L’id du banquier qui s’occupe du compte pour savoir à quelle banquier la banque devra faire passer la demande du client.

***ClientActor :***

La classe ClientActor correspondra à un client et aura pour but d’envoyer des demandes d’ajout ou de retrait à la banque.

Un client pourra envoyer trois types de message à la banque :

* Un message de connexion qu’il enverra à la banque afin d’obtenir le solde actuel de son compte. Se message devra prendre en paramètre l’id du client afin que la banque puisse trouver le compte qui lui appartiennent. Il attendra un objet de type Compte en retour.
* Un message d’ajout qui prend en paramètre le montant à ajouter ainsi que le compte du client. Il attendra le nouveau solde du compte en retour.
* Un message de retrait qui prend en paramètre le montant à retirer ainsi que le compte du client sur lequel il souhaite retirer de l’argent. Il attendra le nouveau solde du compte en retour.

Un client connait naturellement la banque à laquelle il appartient et à qui il doit communiquer, il aura donc un attribut correspondant à cette dernière. Il devra aussi posséder un attribut id afin que la banque puisse retrouver son compte lorsqu’il lui enverra un message de connexion. Pour finir il devra aussi avoir un attribut de type compte qui correspondra naturellement à son compte.

***Acteur BanqueActor :***

La classe BanqueActor à pour but de faire le lien entre les clients et les banquiers qui gère les différents comptes.

Elle n’enverra pas de message en particulier mais pourra réceptionner différents messages provenant des clients.

* Elle pourra réceptionner un message de connexion auquel il renverra le solde actuel du compte qu’elle aura récupérer depuis la base de données.
* Si elle reçoit un message d’ajout ou de retrait, elle redirigera le message vers le banquier en charge du compte.

L’acteur banque sera celui qui créera les différents acteurs banquier et persistance ca ses derniers sont directement lié à cette dernière.

***Acteur BanquierActor :***

Les banquiers seront les acteurs qui effectueront les modifications sur le compte après avoir reçu le message du client.

Lorsqu’un banquier recevra un message d’ajout ou de retrait d’un client qui lui aura été transmis par la banque, il effectuera la modification sur le compte et enverra un message aux PersistancesActor pour qu’ils enregistrent la modification dans la base de données. Il répondra aussi en client en lui envoyant le nouveau solde du compte.

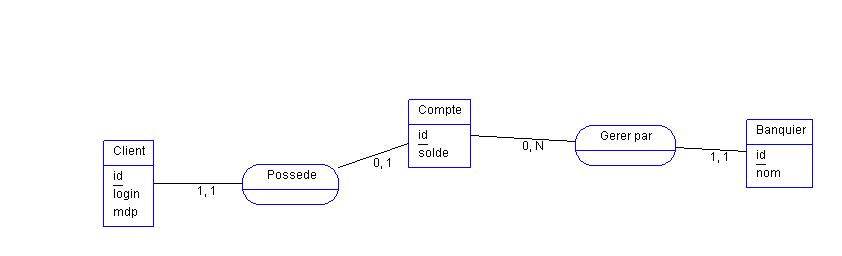
***Acteur PersistanceActor :***

Le routage sera utilisé pour créer les acteurs de persistances afin qu’il puisse se répartir la charge de travail qu’est l’enregistrement des modifications dans la base de données. L’enregistrement en base de données étant couteux, avoir un pool acteurs spécifiques qui se partage cette tache permettra d’accélérer le processus s’il y a de nombreuse demande.

Les acteurs de persistances non qu’un seul but et ne recevront qu’un type de message provenant d’un banquier leurs demandant d’enregistrer le solde du compte après modifications dans la base de données.

***La base de données :***

Comme ce projet nécessite d’utiliser un SGBD afin de pouvoir récupérer les informations des comptes en cas d’arrêt systèmes, il est important de bien désigner la base de données afin que l’enregistrement et la récupération d’informations soit efficace.



Ce qui donnera au finale les tables :

**Client** (id INT, login VARCHAR, mot de passe VARCHAR)

**Compte** (id INT, solde INT, #client INT, #banquier INT)

**Banquier** (id INT, nom VARCHAR)

# Architecture logicielle détaillée

# Evaluation des performances

# Comment compiler le code

# Jeu de test