Programmation WEB

Initiation à Spring avec Spring-boot

Le fichier est susceptible d'être mis à jour durant le module. Vous trouverez ces mises-à-jour dans le repository du projet (https://github.com/Kaway/my-small-bank-base – Lien direct vers le PDF :)

Mises-à-jour :

- 2021-02-10 17:10 – Ajout des règles de validation et des points d'attention pour la notation
- 2021-02-02 21:40 – Version initiale

Table of Contents

Introduction

Le but de ce module est d'apprendre à utiliser Spring-boot au travers de la création d'une mini banque en ligne aux fonctionnalités réduites.

La banque permettra de créer des clients, des comptes, d'associer les comptes aux clients et de faire des transferts d'argent entre différents comptes, le tout à partir de requêtes **HTTP REST**. Selon le temps disponibles, on pourra ajouter des nouvelles fonctionnalités, comme recevoir des chèques, ou associer une carte bancaires.

Le projet est découpé en plusieurs package qui correspondent globalement à un ensemble de fonctionnalité ; par exemple la gestion des comptes et la gestion des transferts. Ensuite, chaque fonctionnalité (*domain*) est découpé en couches :

- **exposition**, pour la gestion des requêtes HTTP et des **DTOs** vers et depuis le client

- **domain**, pour les règles métiers et nos objets du modèles
- **infrastructure**, pour la persistance des données (insertion en base)

La couche domain ne manipulera que des objets du *model*, en aucun cas les **DTOs** et **Entities**, qui devront être converties en objet du *model* avant d'arriver dans le domain.
L'infrastructure sera responsable de l'insertion en base de données, et de la transformations entre nos objets du model et nos JPA Entities (**@Entity**)
L'exposition sera responsable des communications entre le client HTTP (Postman, Insomnia, navigateur web) ou le serveur (à travers les controllers **@RestController**) ; avant d'appeler des méthodes du service, les **DTOs** seront convertis en objet du model.

Fonctionnalités de base

- Pour les clients de la banque (Holder)
  - Créer un client (**POST** sur /holders)
  - Récupérer les informations d'un client (avec la liste de ses comptes) (**GET** sur /holders/{id})
  - Récupérer les informations de tous les clients (**GET** sur /holders avec la liste de leurs comptes)
  - Changer l'adresse d'un client (**PUT** sur /holders/{id}/address)
  - Changer les informations d'un client (**PATCH** sur /holders/{id}, on ne transmet que les champs qui doivent changer)
- Pour les comptes (Account)
  - Créer un compte (**POST** sur /accounts, en précisant l'ID du Holder)
  - Récupérer la liste des comptes (**GET** sur /accounts, en incluant les informations du Holder – nom, prénom)
  - Récupérer les informations d'un compte (**GET** sur /accounts/{id})
  - Supprimer un compte (**DELETE** sur /accounts/{id}) ; si le compte a un solde inférieur à 0, la suppression échoue ; si le compte est d'un type dont le solde minimum est supérieur à 0, on indiquera dans le body de la requête l'id du compte vers lequel transférer le solde ;
- Pour les transferts (Transfer)
  - Créer un transfert (**POST** sur /transfers) ;
  - Récupérer la liste de tous les transferts triés par date d'exécution (**GET** sur /transfers avec un «query parameter» – @RequestParam à ASC ou DESC selon le sens du tri)

Fonctionnalités supplémentaires

- Pour les comptes
  - Calculer la rémunération de tous les comptes en fonction du taux d'intérêt du compte (**POST** sur /accounts/compute-interest)

    - Création d'une carte liée à un compte (**POST** sur /accounts/{id}/cards)
    - Récupération de la liste des opérations ; une opération est un transfert, un paiement par carte ou le versement d'intérêts (**GET** sur /accounts/{id}/operations)
- Pour cartes (Cards)
    - Effectuer un paiement par carte (**POST** sur /cards/{id})
    - Bloquer une carte – interdire les paiements depuis la carte (**PUT** sur /cards/{id}/lock)
    - Débloquer une carte (**DELETE** sur /cards/{id}/lock)

Règles de validation

- création et modification d'un **Holder** :
    - tous les champs doivent avoir une valeur (non nulle et non vide)
    - le **holder** doit avoir 18 ans
- création d'un **Account** :
    - **holder.id** doit être non nul et non vide
    - **holder.id** doit être l'ID d'un Holder existant
    - **balance** doit être supérieur à 0 et supérieure au minimum autorisé selon le type de compte (voir Category)
- création d'un **Transfer**
    - les comptes **from** et **to** doivent exister
    - le **montant** minimum est 1
    - le compte **from** doit avoir assez de fonds (à la fin, doit rester le minimum autorisé selon le type de compte – voir **Category**)
    -

N'hésitez pas à rajouter vos propres règles (si elles sont pertinentes).

Autres

Livrables – par groupe de 3 maximum ;

- URL du repository public (Github, Gitlab, Bitbucket)
  - Le code Java (si possible qui compile, et s'exécute)
  - Les scripts SQL (Postgres ou MySQL)
  - Une collection Postman ou Insomnia avec les requêtes manquantes)
  - Le README.md avec les noms et prénoms des membres du groupe
- Hash du commit

Points d'attentions pour la notation :

- Paths des endpoints de vos controllers (les bons verbes HTTP, avec les bonnes URLs)
- Présences des clés étrangères dans les scripts SQL
- Lisibilité du code (on n'imbrique pas 3 appels de méthodes sur une seule ligne)
- Pas de mélange entre DTO, Entity et Model
- Visibilité des méthodes (private, public et protected si besoin)
- Utilisation des streams et des optionals aux bon endroits