Blanquart Julien
Chalet Lionel
Djenane Anthony
Masquelier Thomas

Rapport de Projet IFI

Introduction

Dans le cadre de ce projet, il nous a été demandé d'implémenter un distributeur de banque. Ce distributeur doit fonctionner de consort avec des projets banques qui fournissent une API. Ces derniers sont implémentés par d'autres étudiants.

Nous avons décidé d'implémenter les fonctionnalités suivantes pour notre distributeur :

- S'authentifier
- Afficher son solde
- Retirer de l'argent
- Effectuer un virement
- Mettre fin à sa session

Nous sauvegardons les transactions dans une base de données, afin de garder un historique. Nous avons aussi décidé d'implémenter des fonctionnalités supplémentaires car celles-ci figuraient dans l'API des banques :

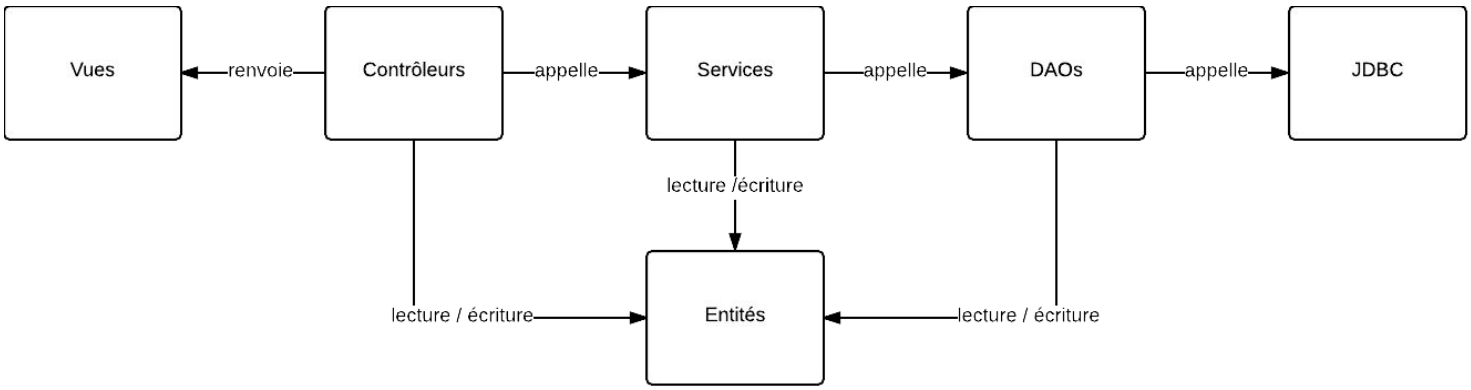
- Créer un compte
- Faire un dépôt d'argent

Ce rapport présente l'architecture du projet et ses choix technologiques.

Architecture

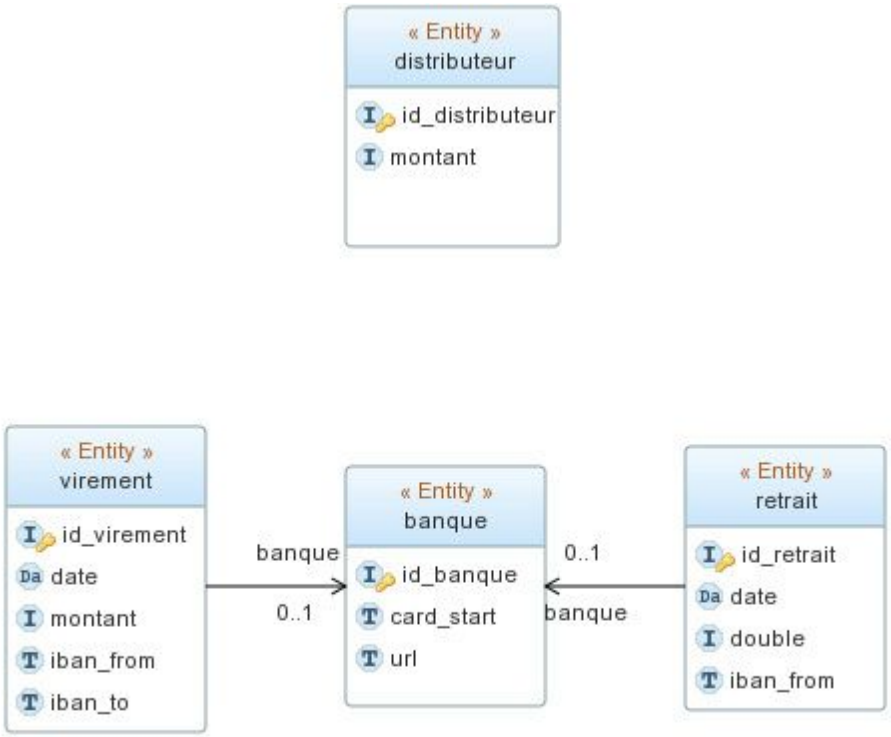
Couches applicatives

L'application est découpées en différentes couches que voici :



- Les **vues** sont des pages html vues par l'utilisateur.
- Les **contrôleurs** reçoivent la requête de l'utilisateur et sont chargées de renvoyer une vue. Cette vue contient certaines informations qui doivent être synchronisées avec les entités. Cela se fait à l'aide de DTO (Data Transfert Objects) que nous n'avons pas implémentés car Thymeleaf (voir choix technologiques) s'en charge pour nous. Les contrôleurs font appel à des services pour effectuer un traitement sur ces entités.
- Les **services** décrivent l'ensemble des fonctionnalités fournies par l'application. Ils se chargent de lire les entités remplies par le contrôleur, d'effectuer un traitement dessus et de mettre à jour le résultat dans la base de données grâce aux DAO (Data Access Objects).
- Les **DAOs** agissent comme le lien entre les entités et les objets de la base de données. Un changement en mémoire n'est en effet pas automatiquement répercuté dans la base de données, d'où leur intérêt.
- Le driver **JDBC** n'est pas implémenté par nos soins mais est utilisé pour l'accès à la base de données.
- Les **entités** sont nos objets métiers. Elles contiennent toutes les données de l'application et correspondent à la couche modèle du framework MVC. Elles sont répliquées dans la base de données.
- Il y a également des classes **utilitaires** et d'**exception** qui sont mises à part pour éviter de surcharger les autres classes. (Celles-ci ne figurent pas sur le schéma)

Schéma de base de données



Nous distinguons 4 entités dans la base de données :

- Le **distributeur** se charge de retenir le montant restant en liquide dans le distributeur.
- Le **virement** se charge de retenir les informations d'un virement dans l'optique de retenir un historique des transactions.
- Le **retrait** se charge de retenir les informations d'un retrait dans l'optique de retenir un historique des transactions.
- La **banque** sert à identifier l'url à laquelle s'adresser en fonction du numéro de carte de l'utilisateur. En effet, le début du numéro de carte identifie la banque qui l'a issue.

Tous les montants sont des entiers exprimés en euros cents.

Choix technologiques

Ce projet est codé en Java. Les dépendances sont gérées à l'aide de Maven.

Nous utilisons le framework Spring Boot qui permet de développer un serveur applicatif. Des alternatives existent telles que EJB ou Spring, mais notre choix s'est porté sur Spring pour sa facilité d'utilisation. En effet ce dernier s'affranchit des fichiers de configuration à n'en plus finir et va même jusqu'à intégrer son propre serveur Tomcat qui s'exécute en tant que simple application Java.

Nous utilisons une base de données PostgreSQL car chacun des membres du groupe la connait bien.

Pour gérer la persistence, nous étions d'abord partis sur le framework Hibernate. Celui-ci est très complet et permet même de générer entièrement sa base de données à partir d'objets Java. Cela se fait hélas par le biais de nombreuses annotations. De plus le processus d'interaction avec la base de données se fait à l'aide soit d'un EntityManager soit d'une Hibernate Session. Les deux étant d'une utilisation complexe, nous avons cherché une solution plus simple. MyBatis s'est montré être le remplaçant idéal. Il ne permet pas de générer la base de données à l'aide d'annotations comme Hibernate mais simplement de mapper des objets Java à des entités dans la base de données. L'interaction avec la base de données est rendue très simple puisqu'il suffit d'associer une méthode Java à une requête SQL telle un select, un update, un insert ou un delete.

Pour les vues, nous étions au départ partis pour utiliser des JSP. Cependant, après avoir découvert Thymeleaf, nous nous sommes orientés vers celui-ci. Il permet par exemple de lier facilement un objet Java à un formulaire dans la vue, nous simplifiant la tâche par la même occasion.

Répartition des tâches

Au début du projet, nous avions fait une première répartition des tâches :

- Julien devait se charger des DAO
- Lionel et Thomas devaient se charger des contrôleurs
- Anthony devait se charger des JSP

Au final, tout le monde a touché à toutes les parties du projet tout en se concentrant malgré tout sur son son affectation initiale. Cela a au final donné la répartition suivante :

Julien

- Entités et création de la base de données
- DAO : implémentation initiale sous Hibernate
- Services
    - Création de la méthode utilitaire pour faire une requête à la banque
    - Implémentation d'une partie de InteractionBanque
    - Implémentation de InteractionDistributeur
- Javadoc
- Rapport
    - Introduction
    - Ebauche de la répartition des tâches

Lionel

- Schéma de base de données
- DAO : remplacement de Hibernate par MyBatis
- Contrôleurs
- Services : Implémentation d'une partie de InteractionBanque
- Gestion des exceptions
- Corrections de bugs dans toutes les couches
- Rapport : tout le reste

Thomas

- Script pour recréer la base de données
- DAO : remplacement de Hibernate par MyBatis
- Contrôleurs
- Pages HTML
- Services
    - Création de la méthode de hash
    - Amélioration de la méthode utilitaire pour faire une requête à la banque
    - Implémentation d'une partie de InteractionBanque
- Corrections de bugs dans toutes les couches

Anthony

- Contrôleurs
- Pages HTML
- CSS
- Rapport
    - Ebauche des choix technologiques
    - Conclusion

Utilisation

- `git pull https://github.com/u531355/tiir_distrib.git`
- `mvn spring-boot:run`

L'application sera lancée en local sur le port 8080.

Conclusion

Ce projet nous a permis de découvrir le framework Spring, mais aussi Thymeleaf et MyBatis, qui peuvent être utilisés conjointement afin de réaliser une application web en Java. Nous avons pu également nous familiariser plus en détail avec le patron de conception MVC, dont nous nous sommes servis pour créer l'architecture de notre application web.