Rapport SimpleCash

Vincent Bensadi

Novembre 2025

Contents

1 Introduction **2**

2 User stories minimales **3**

3 Bilan du projet **4**

4 API REST : routes et utilisation **5**

5 Références **7**

1 Introduction

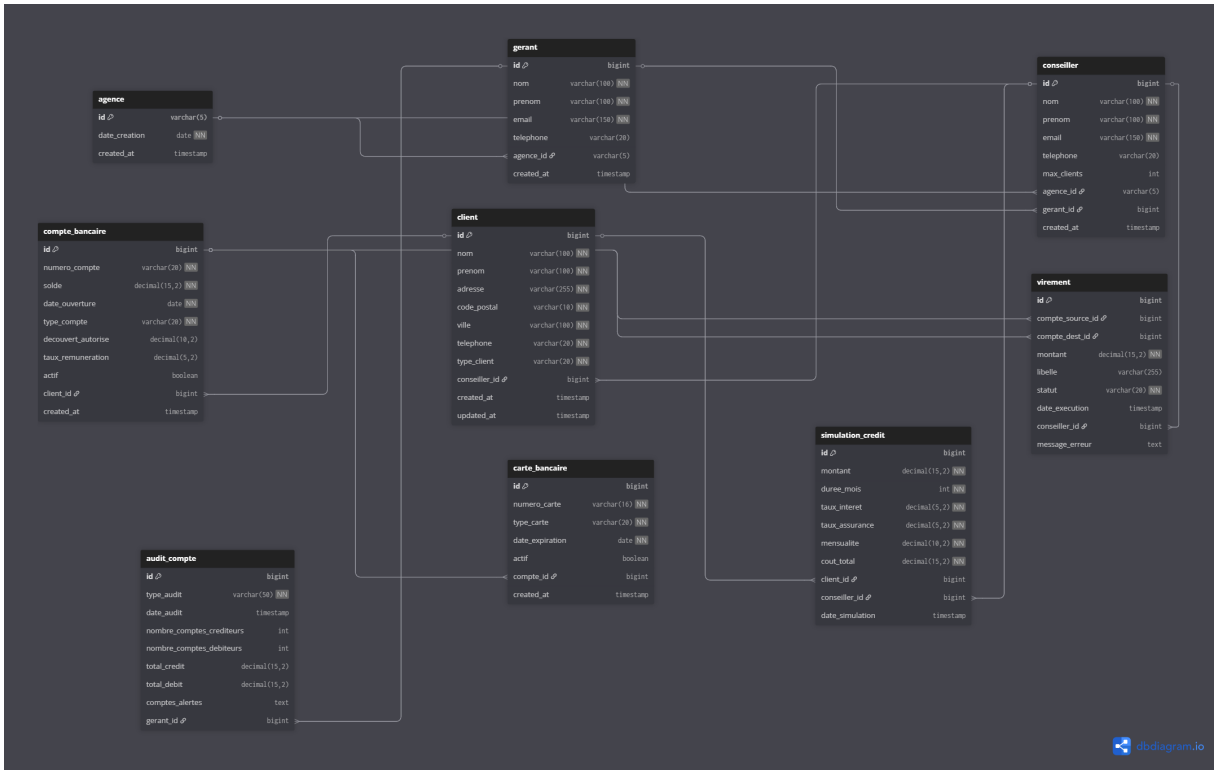

Figure 1: UML - Diagramme de base de données

Les diagrammes UML ci-dessus illustrent les principales entités du SI SimpleCash : Client, Conseiller, Gerant, Agence, Compte, CarteBancaire, ainsi que leurs relations.

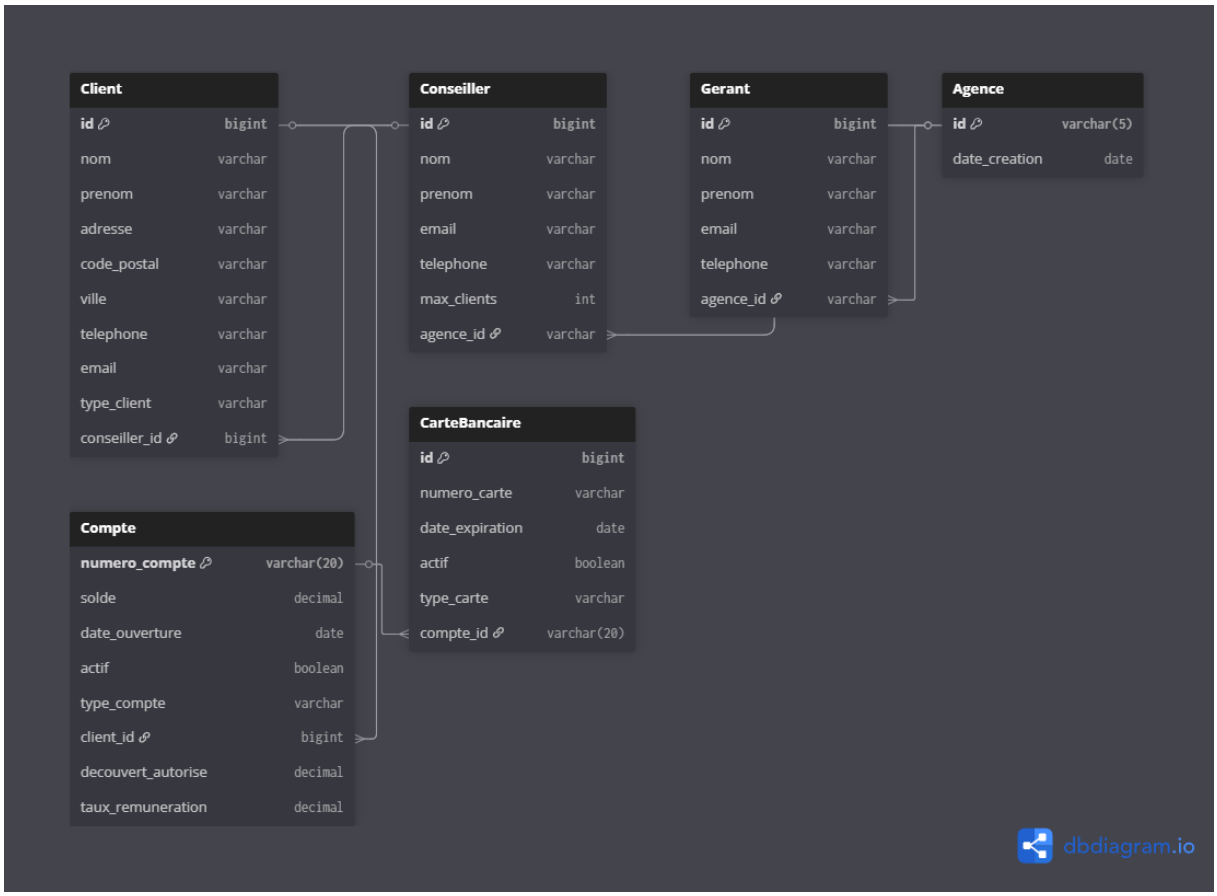

Figure 2: UML - Diagramme de classes

2 User stories minimales

Les user stories sont présentes dans les issues du dépôt GitHub du projet. Elles couvrent notamment :

- La création, consultation, modification et suppression d'un client
- La gestion des comptes (crédit, débit, virement)
- L'audit des comptes
- La traçabilité des opérations sensibles

Priorisation des issues

Chaque issue est priorisée avec un label : `low`, `medium`, `high` selon l'urgence et l'importance métier. Cela a permis d'organiser le développement et de livrer les fonctionnalités essentielles en premier.

3 Bilan du projet

Fonctionnalités implémentées

- Gestion complète des clients (CRUD)
- Gestion des comptes bancaires (crédit, débit, virement)
- Audit des comptes et détection des dépassements
- Désactivation des cartes bancaires lors de la suppression d'un client
- Documentation JavaDoc

Reste à faire

- Gestion avancée des conseillers et de leur rattachement aux agences
- Interface de gestion des agences
- Sécurité et authentification des utilisateurs
- Gestion des rôles (gérant, conseiller, client)
- Interface utilisateur front-end

Difficultés rencontrées et contournements

- **Conceptualisation et modélisation** : Beaucoup de temps consacré à la modélisation des concepts métier pour garantir la cohérence des relations et des règles métier. Les choix d'architecture ont été validés avec des diagrammes UML et des échanges réguliers.
- **Gestion des cascades** : La suppression d'un client implique la suppression en cascade de ses comptes et la désactivation de ses cartes bancaires. Cela a nécessité une bonne maîtrise de JPA et des annotations de mapping.
- **Traçabilité et auditabilité** : L'utilisation de Spring AOP pour tracer les opérations sensibles (virements) dans un fichier de log dédié a permis de garantir l'auditabilité du SI.
- **Difficultés techniques** : Problèmes de dépendances Maven (Swagger), configuration de Lombok et mapping JPA, résolus par nettoyage du cache, documentation et échanges avec la communauté.

4 API REST : routes et utilisation

Ce document est à remettre au client pour validation et suivi du projet.

Liste des routes principales

- **Client**
  - `POST /api/clients` : Créer un client (body JSON, retourne HTTP 201)
  - `GET /api/clients/{id}` : Consulter un client (retourne HTTP 200 ou 404)
  - `PUT /api/clients/{id}` : Modifier un client (body JSON, retourne HTTP 200)
  - `DELETE /api/clients/{id}` : Supprimer un client (cascade comptes/cartes, retourne HTTP 204)
- **Compte**
  - `POST /api/comptes/{id}/crediter` : Créditer un compte (body : montant, retourne HTTP 200)
  - `POST /api/comptes/{id}/debiter` : Débiter un compte (body : montant, retourne HTTP 200 ou 400)
- **Virement**
  - `POST /api/virements` : Virement entre comptes (body : source, destination, montant, retourne HTTP 200)
- **Audit**
  - `GET /api/audits/comptes` : Audit des comptes (retourne rapport JSON, HTTP 200)

Comment utiliser et tester l'API

1. Toutes les routes sont testables via le plugin RESTer sur Firefox..
2. **Validation** : Vérifiez les codes retour (201, 200, 204, 400, 404) et les bodies JSON pour chaque opération.
3. **Audit et logs** : Les opérations sensibles (virements) sont tracées automatiquement dans le fichier `virements.log` pour auditabilité.

Exemples de tests

- **Créer un client** :

```
POST /api/clients
Body : {
  "nom": "Dupont",
  "prenom": "Jean",
```

```
  "email": "jean.dupont@example.com",
  "telephone": "0601020304",
  "adresse": "10 rue de la Paix",
  "codePostal": "75001",
  "ville": "Paris",
  "typeClient": "PARTICULIER",
  "conseillerId": 1
}
```

- **Virement** :

```
POST /api/virements
Body : {
  "compteSourceId": "CC001",
  "compteDestId": "CE001",
  "montant": 500.00
}
```

- **Audit** :

```
GET /api/audits/comptes
```

Remarques

- Toutes les routes sont testables via le plugin RESTer sur Firefox.
- Les validations métier sont automatiques (ex : quota conseiller, découvert, cascade suppression).
- Les logs de virement sont consultables pour audit.

5 Références

- **Dépôt GitHub** : `https://github.com/vbensadi/projet-bensadi-vincent`
- **Issues** : Les user stories et tâches restantes sont suivies dans les issues du dépôt.