Objet : Réflexion technique sur la migration SQLite → MongoDB pour la version 2 de la plateforme SaaS de rapprochement bancaire

Contexte:

Dans le cadre du passage à l'échelle de la version 2 de la plateforme SaaS de rapprochement bancaire, il est prévu de remplacer SQLite (base de données utilisée dans la version 1) par MongoDB. Cette migration ne consiste pas en un simple transfert de données, **mais** en une refonte complète du modèle de données, **passant** d'un modèle relationnel à un modèle document, adapté aux besoins de **scalabilité**, de **flexibilité** et de **performance** d'une solution multi-tenant.

1. Objectifs de la migration

- **Scalabilité horizontale :** Support de plusieurs milliers de clients (tenants) avec isolation stricte des données.
- **Flexibilité du schéma :** Adaptation rapide aux évolutions métier sans migrations lourdes.
- **Performance en lecture/écriture :** Optimisation des requêtes fréquentes (transactions, matching, rapports).
- **Intégration native avec Node.js :** Utilisation de Mongoose pour une modélisation robuste et une validation au niveau applicatif.

2. Analyse du schéma SQLite existant

Avant toute migration, une analyse approfondie du schéma SQLite actuel est indispensable :

- ➔ **Tables principales :** users, transactions, invoices, matches, rules, quotas, etc.
- ➔ **Relations et clés étrangères :** Relations 1-N, N-N, contraintes d'intégrité.
- ➔ **Types de données et précision :** Notamment les montants, dates, références bancaires.

3. Stratégie de modélisation MongoDB

MongoDB permet **deux approches** principales :

a) Données embarquées (Embedded)

Idéal pour les données souvent lues ensemble et à faible volumétrie indépendante.

Exemple pour une facture avec ses lignes :

```
{
  "_id": "inv_001",
  "tenantId": "t_123",
  "invoiceNumber": "F2026-001",
  "total": 1500,
  "currency": "EUR",
```

```
  "lines": [
    { "label": "Service Conseil", "amount": 1200 },
    { "label": "TVA 20%", "amount": 300 }
  ],
  "createdAt": "2026-01-28T10:00:00Z"
}
```

b) Données référencées (Referenced)

À privilégier pour les entités volumineuses, partagées ou à cycle de vie indépendant.

Exemple pour les utilisateurs :

```
{
  "_id": "u_456",
  "email": "comptable@entreprise.com",
  "role": "ACCOUNTANT",
  "tenantId": "t_123"
}
```

4. Mapping SQLite → MongoDB

5. Processus de migration proposé

Phase 1 : Audit et extraction SQLite

- ★ Exporter les données au format JSON/CSV via des scripts Node.js ou Python.
- ★ Documenter les relations et contraintes métier.

Phase 2 : Transformation des données

- ★ Regrouper les lignes enfants dans les documents parents (ex: invoice_lines dans invoices).
- ★ Ajouter le champ tenantId à chaque document pour l'isolation multi-tenant.
- ★ Convertir les dates, normaliser les montants et les devises.

Phase 3 : Import dans MongoDB

- ★ Utilisation de mongoimport ou d'un script d'insertion massif (bulk insert).
- ★ Validation des index (tenantId, date, status).

Phase 4 : Vérification et tests

- ★ Comparer les comptages et les totaux financiers.
- ★ Tester les requêtes critiques (matching, reporting, filtres).
- ★ S'assurer de l'absence de données orphelines.

Phase 5 : Adaptation du backend

- ★ Remplacer les requêtes SQL par des requêtes MongoDB (Mongoose).
- ★ Réécrire les jointures en **$lookup** ou repenser le modèle pour les éviter.

★ Mettre à jour la logique métier autour des transactions et du matching.

6. Points de vigilance

★ **Cohérence des données :** MongoDB **n'applique pas de clés étrangères**. La validation doit être gérée au niveau applicatif (Mongoose hooks, validation côté API).

★ **Transactions financières :** Pour les opérations critiques (débit/crédit de quotas, validation de matches), utiliser les transactions multi-documents de MongoDB.

★ **Performance des agrégations :** Bien structurer les documents pour **limiter** les **$lookup coûteux**.

★ **Backup et restauration :** Mettre en place des stratégies de backup régulières (snapshots, mongodump).

7. Recommandations techniques

★ **Base de données :** MongoDB Atlas (managed) pour la scalabilité et la haute disponibilité.

★ **ODM :** Mongoose pour la modélisation, les hooks de validation et le support des transactions.

★ **Indexation :** Index composés sur tenantId + date, tenantId + status, tenantId + bankReference.

★ **Sharding :** À prévoir à long terme pour séparer les données par tenant ou par période.

Conclusion

La **migration SQLite → MongoDB** représente une opportunité stratégique pour aligner l'architecture technique avec les ambitions de scalabilité et de modularité de la version 2. Cette refonte doit être menée avec rigueur, en accord avec les contraintes métier du domaine bancaire et comptable.

Statut : Document de travail – Version 1.0