Compte-rendu du Projet Base de Données - Baie-Électronique

Ensimag 2ème année - 2024-2025

Table des matières

1 Introduction

Ce document constitue le compte-rendu du projet **Baie-Électronique**, visant à concevoir et implémenter une base de données relationnelle pour gérer les ventes aux enchères.

2 Analyse du problème

2.1 Dépendances Fonctionnelles

Voici les dépendances fonctionnelles identifiées et leurs justifications :

- Nom_categorie → description.
  Chaque catégorie de produit est identifiée par son nom unique et possède une description spécifique.
- ID_salle_vente → Nom_categorie.
  Chaque Salle de Vente est identifiée de manière unique et est associée à une seule catégorie de produits.
- ID_produit → nom, prix_revient, Nom_categorie, stock.
  Un produit est identifié de façon unique par son ID_produit, et chaque produit est décrit par son prix de revient, sa catégorie et le stock disponible.
- (ID_produit, Nom_Caractéristique) → valeur.
  Les caractéristiques d'un produit sont définies par la combinaison de son identifiant et du nom de la caractéristique, associées à une valeur spécifique.
- mail → nom, prenom, adresse.
  Chaque utilisateur est identifié de manière unique par son email et possède un nom, un prénom ainsi qu'une adresse postale.
- ID_Vente → montante, limite_offre, revocable, prix_depart, date_fin, prix_descente, Stock_lot, ID_salle_vente, ID_produit, mail.
  Chaque vente est identifiée de manière unique par son ID_Vente et inclut des attributs tels que le type de vente (montante ou descendante), les contraintes sur les offres, les prix de départ et de descente, la date de fin, le stock disponible, ainsi que les références à la salle de vente, au produit et à l'utilisateur ayant initié la vente.
- ID_offre → mail, ID_Vente, date_offre, prix_achat, quantite.
  Chaque offre est identifiée de manière unique par son ID_offre et est associée à un utilisateur, une vente spécifique, une date et heure de dépôt, un prix d'achat proposé ainsi qu'une quantité de produit concernée.

Ces dépendances fonctionnelles structurent le système et garantissent la normalisation des relations.

2.2 Formes normales

Categorie
Clé primaire : `Nom_categorie`
Dépendances Fonctionnelles :

$$\text{Nom\_categorie} \rightarrow \text{description}$$

La table Catégorie respecte la Première Forme Normale (1FN) car toutes les valeurs des attributs sont atomiques et non Null. Elle est en Deuxième Forme Normale (2FN) car `description` dépend entièrement de la clé primaire simple `Nom_categorie`, sans dépendances partielles. Enfin, elle est en Troisième Forme Normale (3FN) car il n'existe aucune dépendance transitive ;

Salle_de_vente
Clé primaire : `ID_salle_vente`
Dépendances Fonctionnelles :

$$\text{ID\_salle\_vente} \rightarrow \text{Nom\_categorie}$$

La table `Salle_de_vente` est conforme à la 1FN puisque toutes les valeurs des attributs sont atomiques et non Null. Elle satisfait la 2FN car `Nom_categorie` dépend entièrement de la clé primaire simple `ID_salle_vente`. De plus, aucune dépendance transitive n'est présente, garantissant ainsi la conformité à la 3FN.

Produit
Clé primaire : `ID_produit`
Dépendances Fonctionnelles :

$$\text{ID\_produit} \rightarrow \text{prix\_revient}, \text{Nom\_categorie}, \text{stock}, \text{nom}$$

La table `Produit` respecte la 1FN avec des valeurs atomiques et non NULL. Elle est en 2FN car tous les attributs non clés (`prix_revient`, `Nom_categorie`, `stock`) dépendent entièrement de la clé primaire simple `ID_produit`. En outre, aucune dépendance transitive n'existe, ce qui assure la conformité à la 3FN.

Caracteristique
Clé primaire : `(ID_produit, Nom_Caracteristique)`
Dépendances Fonctionnelles :

$$(\text{ID\_produit}, \text{Nom\_Caracteristique}) \rightarrow \text{valeur}$$

La table `Caracteristique` est en 1FN grâce à des valeurs atomiques et non Null. Elle respecte la 2FN car l'attribut `valeur` dépend entièrement de la clé primaire. De plus, aucune dépendance transitive n'est présente, assurant ainsi la 3FN.

Utilisateur
Clé primaire : `mail`
Dépendances Fonctionnelles :

$$\text{mail} \rightarrow \text{nom}, \text{prenom}, \text{adresse}$$

La table `Utilisateur` respecte la 1FN avec des valeurs atomiques et non Null. Elle est en 2FN car tous les attributs non clés (`nom`, `prenom`, `adresse`) dépendent entièrement de la clé primaire simple `mail`. En outre, aucune dépendance transitive n'existe, garantissant ainsi la 3FN.

Vente
Clé primaire : `ID_Vente`
Dépendances Fonctionnelles : Dépendances Fonctionnelles :

$$\begin{aligned}\text{ID\_Vente} \rightarrow\ & \text{montante},\ \text{limite\_offre},\ \text{revocable},\ \text{prix\_depart},\\ & \text{date\_fin},\ \text{prix\_descente},\ \text{Stock\_lot},\\ & \text{ID\_salle\_vente},\ \text{ID\_produit},\ \text{mail}.\end{aligned}$$

La table `Vente` est conforme à la 1FN car toutes les valeurs des attributs sont atomiques et non Null. Elle satisfait la 2FN puisque tous les attributs non clés dépendent entièrement de la clé primaire simple `ID_Vente`. Pour la 3FN, bien que des clés étrangères (`ID_salle_vente`, `ID_produit`, `mail`) soient présentes, elles ne déterminent pas d'autres attributs au sein de la table `Vente`.

Offre
Clé primaire : `ID_offre`
Dépendances Fonctionnelles :

$$\text{ID\_offre} \rightarrow \text{mail}, \text{ID\_Vente}, \text{date\_offre}, \text{prix\_achat}, \text{quantite}$$

La table `Offre` respecte la 1FN avec des valeurs atomiques et non Null. Elle est en 2FN car tous les attributs non clés dépendent entièrement de la clé primaire simple `ID_offre`. Aucune dépendance transitive n'existe, assurant ainsi la conformité à la 3FN.

Conclusion : Toutes les tables de la base de données respectent les trois premières formes normales (1FN, 2FN, 3FN). Les dépendances fonctionnelles sont correctement définies, éliminant les redondances et assurant l'intégrité des données. Cela garantit un modèle robuste, cohérent et aligné avec les besoins métier de l'application de ventes aux enchères de la société Baie-électronique.

2.3 Contraintes de valeur

- Une catégorie est définie par un **nom unique** et une **description**.
- Les prix des produits doivent être **supérieurs ou égaux à 0**.
- Le stock d'un produit est une valeur entière **positive**.
- La quantité d'une offre doit être **comprise entre 1 et le stock disponible** pour le produit correspondant.

2.4 Contraintes de multiplicité

- Un produit appartient à **une seule catégorie**.
- Une catégorie peut contenir **plusieurs produits**.
- Une salle de vente contient **plusieurs ventes** (au moins une).
- Une vente est associée à **un seul produit** et à **une seule salle de vente**.
- Une vente peut être associée à **plusieurs offres**.
- Une offre peut concerner **un sous-lot spécifique** ou **un produit complet**.

2.5 Contraintes contextuelles

- Une offre est invalide si elle ne respecte pas le **sens de la vente** (montante ou descendante) :
  - Pour une vente montante (`montante = TRUE`), le prix proposé doit être **supérieur au prix de départ** ou à l'offre précédente.
  - Pour une vente descendante (`montante = FALSE`), le prix proposé doit être **inférieur ou égal au prix courant**.
- Une vente peut être annulée si le **prix de revient du produit** n'est pas atteint (dans les ventes révocables `revocable = TRUE`).
- Si `limite_offre = TRUE`, un utilisateur ne peut soumettre qu'**une seule offre** pour cette vente.
- Par défaut, les ventes sont :
  - Montantes (`montante = TRUE`),
  - Non révocables (`revocable = FALSE`),
  - Sans limitation de temps (`date_fin = NULL`),
  - Et permettent **plusieurs offres par utilisateur** (`limite_offre = FALSE`).
- Si une **date de fin** (`date_fin`) est définie, elle doit être **supérieure à la date actuelle**.
- Le **prix proposé** pour une offre doit être **strictement supérieur au prix de départ**.

3 Conception UML (Entités/Associations)

3.1 Schéma UML

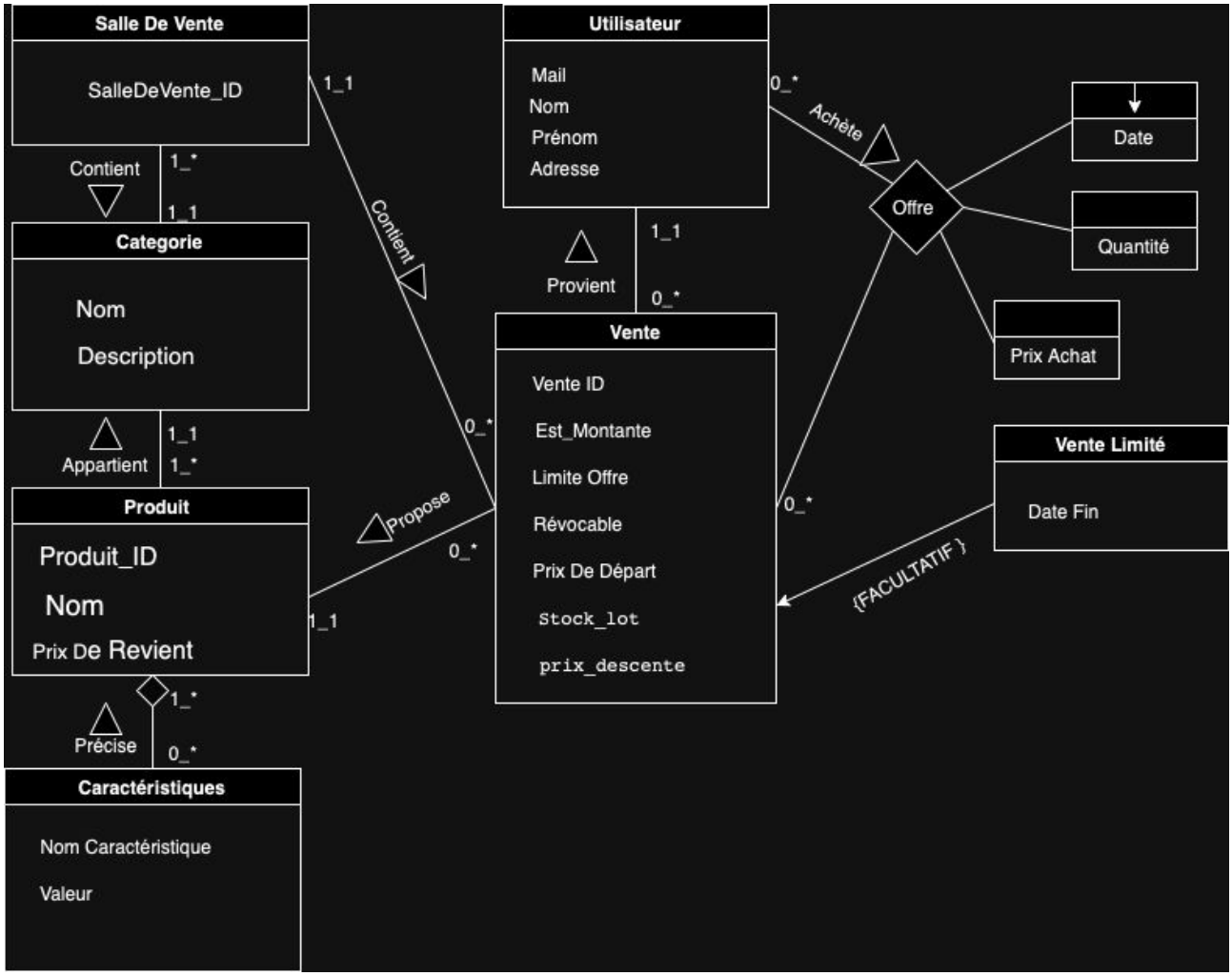

FIGURE 1 – Schéma UML des entités et associations du projet

3.2 Explication des choix

— Les entités (`Utilisateur`, `Produit`, `SalleDeVente`, `Vente`, `Categorie`) sont définies en fonction de la description du problème.
— Les associations sont modélisées pour respecter les contraintes de multiplicité.
— Les attributs tels que les prix et les dates sont typés pour assurer la cohérence des données.

4 passage du modèle Entité/Association vers un schéma Relationnel

4.1 Relations

- Relation **Utilisateur** : (Email , `Nom`, `Prénom`, `Adresse`).

- Relation **Produit** (IDProduit , `PrixDeRevient`, `nom` , `NomCatégorie` FK).
  Explication :
  — La relation `Produit` est une entité faible par rapport à `Catégorie` dans l'association `Produit-Catégorie`.
  — Pour respecter cette contrainte, nous ajoutons l'attribut clé primaire de `Catégorie` comme clé étrangère (`NomCatégorie`) dans la table `Produit`.
- Relation **SalleDeVente** (IDSalleVente , `NomCatégorie` FK).
  Explication :
  — La relation `SallDeVente` est une entité faible par rapport à `Catégorie` dans l'association `SalleDeVente-Catégorie`.
  — Pour respecter cette contrainte, nous ajoutons l'attribut clé primaire de `Catégorie` comme clé étrangère (`NomCatégorie`) dans la table `SalleDeVente`.
- Relation **Categorie** (NomCatégorie , `description`).

- Relation **Caracteristique** (NomCaracteristique , `valeur` ,`IDProduit`).

- Relation **Vente** (IDVente, `montante`, `limiteOffre`, `revocable`, `prixDepart`, `dateFin`, `prixDescente`, `StockLot`, `IDSalleVente` FK, `IDProduit` FK, `Email` FK).

  Explication :
  — La relation `Vente` est une entité faible par rapport à `Produit`,`SalleDeVente`,`Utilisateur` respectivement dans les associations `Produit-Vente`,`SalleDeVente-Vente` et `Utilisateur-Vente` .
  — Pour respecter cette contrainte, nous ajoutons les clés primaires de `Produit` :IdProduit , `SalleDeVente` :IdSalleVente et `Utilisateur` :Email comme clés étrangères dans la table `SalleDeVente`.
- Relation **Offre** (IdOffre, `Email` FK, `IDVente` FK, `dateOffre`, `prixAchat`, `quantite`).

4.2 Contraintes non implantables en relationnel

— Les contraintes liées à la logique métier, comme l'annulation des ventes révocables.
— Le contrôle du délai maximal entre deux offres (pour les ventes à durée libre).
— Validation de l'ordre des offres dans les ventes montantes (offres toujours croissantes).
— Limitation du nombre d'offres par utilisateur pour certaines ventes.
— Diminution automatique du prix dans les ventes descendantes (prix diminuant toutes les minutes).
— Gestion des sous-lots pour s'assurer que la quantité totale vendue ne dépasse pas le stock disponible.
— Détermination du premier enchérisseur gagnant dans les ventes descendantes.
— Contrôle de la validité temporelle des offres en fonction de l'heure actuelle.
— Respect du type de vente lors des offres (montante ou descendante) et des règles associées.
— L'acceptation d'une offre peut entraîner la suppression de toutes les offres devenues incompatibles.

5 Analyse des fonctionnalités

5.1 Transactions et Requêtes SQL

5.1.1 Mise en place d'une salle de vente

Pour la mise en place d'une salle de vente, nous avons implémenté une méthode en Java qui encapsule la requête SQL correspondante. Cette méthode gère les spécificités des ventes (montante, descendante, révocable, etc.) et assure une gestion robuste des transactions grâce à des contrôles stricts et l'utilisation de *Prepared Statements*. La requête SQL utilisée est la suivante :

```
INSERT INTO Vente (montante, limite_offre, revocable, prix_depart,
date_fin, prix_descente, Stock_lot, ID_salle_vente, ID_produit, mail)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

Voici un extrait de la méthode Java associée :

```
PreparedStatement statement = connection.prepareStatement(SQLRequests.SQL_INSERT_NEW_SAL
statement.setBoolean(1, sale.isIncrease());
statement.setBoolean(2, sale.isLimitedOffers());
statement.setBoolean(3, sale.isCancelable());
statement.setFloat(4, sale.getPrixDepart());
statement.setTimestamp(5, sale instanceof LimitedSale ? ((LimitedSale) sale).getEndDate(
statement.setFloat(6, sale.getPriceDecreaseAmount());
statement.setInt(7, sale.getQuantity());
statement.setInt(8, sale.getSaleRoomId());
statement.setInt(9, sale.getProductId());
statement.setString(10, sale.getOwner());
```

5.1.2 Enchère d'un utilisateur

Lorsqu'un utilisateur soumet une enchère, une transaction vérifie plusieurs conditions essentielles, telles que :

- Si les enchères multiples sont autorisées pour l'utilisateur.
- Si l'enchère respecte la limite de temps.
- Si la quantité demandée ne dépasse pas le stock disponible.

La requête SQL utilisée pour insérer une enchère est la suivante :

```
INSERT INTO Offre (mail, ID_Vente, date_offre, prix_achat, quantite)
VALUES (?, ?, ?, ?, ?);
```

Voici un extrait de la méthode Java qui encapsule cette opération :

```
PreparedStatement statement = connection.prepareStatement(SQLRequests.SQL_INSERT_NEW_OFF
statement.setString(1, offer.getRequest());
statement.setInt(2, offer.getIdSale());
statement.setTimestamp(3, offer.getDate());
statement.setDouble(4, offer.getPrice());
statement.setInt(5, offer.getQuantity());
```

5.1.3 Détermination du gagnant d'une enchère

Pour déterminer le gagnant d'une enchère, nous utilisons une requête SQL qui trie les offres par intérêt, défini comme le ratio prix/quantité. La requête utilisée est la suivante :

```
SELECT * FROM Offre
WHERE ID_Vente = ?
ORDER BY prix_achat / quantite DESC;
```

Cette requête est intégrée dans une méthode Java, qui itère sur les résultats pour déterminer l'utilisateur gagnant et met à jour la base de données en conséquence :

```
PreparedStatement statement = connection.prepareStatement(SQLRequests.SQL_RETRIEVE_OFFER
statement.setInt(1, sale.getId());
ResultSet resultSet = statement.executeQuery();
while (resultSet.next()) {
    Offer offer = new Offer(
        resultSet.getString("mail"),
        resultSet.getInt("ID_offre"),
        resultSet.getTimestamp("date_offre"),
        resultSet.getDouble("prix_achat"),
        resultSet.getInt("quantite"),
        resultSet.getInt("ID_Vente")
    );
    offers.add(offer);
}
```

En cas d'offres incompatibles (quantité supérieure au stock disponible), celles-ci sont supprimées :

```
DELETE FROM Offre
WHERE ID_Vente = ? AND quantite > (SELECT Stock_lot FROM Vente WHERE ID_Vente = ?);
```

5.1.4 Gestion des offres : acceptation et refus

Deux processus principaux sont implémentés pour gérer les offres dans le système : l'acceptation et le refus.

Acceptation d'une offre Lorsqu'une offre est acceptée, les étapes suivantes sont effectuées :

- Vérification de l'existence de l'offre : Une requête `SELECT` est exécutée pour s'assurer que l'offre spécifiée existe.
- Mise à jour du stock : Le stock correspondant à la vente est décrémenté selon la quantité de l'offre acceptée.
- Suppression des offres incompatibles : Les offres dépassant le stock restant sont supprimées.
- Suppression de la vente : Si le stock atteint zéro, la vente est automatiquement supprimée.

Ces opérations sont encapsulées dans une transaction, avec un `ROLLBACK` en cas d'erreur ou d'incohérence, et un `COMMIT` uniquement après la validation de toutes les étapes.

Refus d'une offre Pour refuser une offre, une simple suppression est effectuée :

```
DELETE FROM Offre WHERE ID_offre = ?;
```

Cette opération est encapsulée dans une transaction pour garantir sa cohérence avec le reste des données.

5.1.5 Gestion des erreurs et transactions sécurisées

Toutes les opérations sont encapsulées dans des transactions utilisant explicitement des *commit* et *rollback* pour garantir la cohérence des données :

```
try {
    connection.commit();
} catch (SQLException e) {
    connection.rollback();
    throw e;
}
```

5.1.6 Requêtes utilitaires

En complément des transactions principales, plusieurs requêtes SQL sont utilisées pour des opérations de support. Par exemple :

— **Récupération des catégories disponibles** :
```
SELECT * FROM Categorie;
```

— **Récupération des salles de vente par catégorie** :
```
SELECT * FROM Salle_de_vente WHERE Nom_categorie = ?;
```

Ces requêtes, bien qu'utilitaires, participent à la robustesse et à la fluidité de l'application.

5.1.7 Choix du niveau d'isolation : Serializable

Dans notre système, nous avons choisi d'utiliser le niveau d'isolation **Serializable** pour toutes les transactions. Ce niveau d'isolation, configuré dans la classe `Main`, est défini dès l'établissement de la connexion avec la base de données. Voici le code correspondant :

```
dbConnection = DriverManager.getConnection(
        SQLRequests.SQL_SERVER_LOCATION, SQLRequests.SQL_USER, SQLRequests.SQL_PASSWORD)
dbConnection.setAutoCommit(false);
dbConnection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
```

Ce choix garantit une cohérence totale des données en évitant les anomalies suivantes :

— **Lectures fantômes (*Phantom Reads*)** : Les nouvelles données ajoutées par d'autres transactions ne sont pas visibles tant que la transaction en cours n'est pas terminée.

— **Lectures non répétables (*Non-Repeatable Reads*)** : Une même requête exécutée plusieurs fois dans une transaction retourne toujours les mêmes résultats.

— **Écritures perdues (*Lost Updates*)** : Deux transactions concurrentes ne peuvent pas écraser mutuellement leurs modifications.

Bien que ce niveau d'isolation impose une surcharge en termes de performance, il est nécessaire dans un système où la concurrence entre utilisateurs est fréquente, comme les enchères. Il garantit que toutes les transactions sont exécutées de manière strictement séquentielle du point de vue de la base de données.

Configuration automatique Le niveau d'isolation `TRANSACTION_SERIALIZABLE` est défini automatiquement dès que l'application démarre, comme montré ci-dessus. Cela garantit que toutes les transactions respectent ce niveau sans nécessiter de configuration supplémentaire dans les méthodes individuelles.

De plus, l'utilisation systématique de *Prepared Statements* protège contre les attaques par injection SQL, renforçant ainsi la sécurité et la robustesse du système.

5.1.8 Autres fonctionnalités gérées dans les transactions

En plus des cas décrits ci-dessus, notre système gère les fonctionnalités suivantes :

— **Mise à jour du stock après une enchère acceptée :** Une fois qu'une enchère est validée, le stock correspondant est mis à jour :

```
UPDATE Vente SET Stock_lot = Stock_lot - ? WHERE ID_Vente = ?;
```

— **Suppression des ventes invalides :** Si une vente atteint un stock nul ou devient invalide, elle est supprimée :

```
DELETE FROM Vente WHERE ID_Vente = ? AND Stock_lot = 0;
```

6 Bilan du projet

6.1 Organisation

Pour mener à bien ce projet, nous avons d'abord travaillé en collaboration pendant les deux premieres semaines, pour analyser le sujet. Ensemble, nous avons identifié les concepts clés et établi le schéma associations-entités ainsi que les dépendances fonctionnelles (DF). Une fois cette phase de conception terminée, nous avons réparti les tâches entre les membres de l'équipe pour optimiser le travail.

Un groupe s'est principalement concentré sur la réalisation du démonstrateur, un autre groupe s'est chargé de l'implémentation en relationnel, et un dernier groupe s'est occupé de la documentation. Tout au long du projet, nous avons maintenu une communication régulière pour partager nos avancées et mettre en commun les différentes parties du travail.

Enfin, une fois les différentes étapes complétées, nous avons collaboré pour préparer les slides de la soutenance, garantissant ainsi une présentation claire et cohérente de l'ensemble de notre projet.

6.2 Problemes rencontrés

Au cours du projet, nous avons rencontré quelques difficultés, notamment dans l'analyse initiale du sujet et dans l'élaboration du schéma entités-associations. Cette phase a nécessité de nombreuses itérations, car certaines ambiguïtés dans le texte ont conduit à des interprétations différentes. Nous avons dû revoir et ajuster le schéma à plusieurs reprises pour garantir qu'il reflète correctement les besoins du système et les dépendances fonctionnelles identifiées.

7 Mode d'emploi du démonstrateur

— Pour lancer cette application, il faut exécuter dans le terminal la commande suivante : ./gradlew run (nous vous recommandons de travailler avec Intellij).

— Suivre les instructions pour insérer des données, gérer des enchères, ou consulter les résultats.

FIGURE 2 – Créer un compte ou connexion

FIGURE 3 – Création de compte (Ajout d'utilisateur)

FIGURE 4 – Identification ou suppression du compte existant

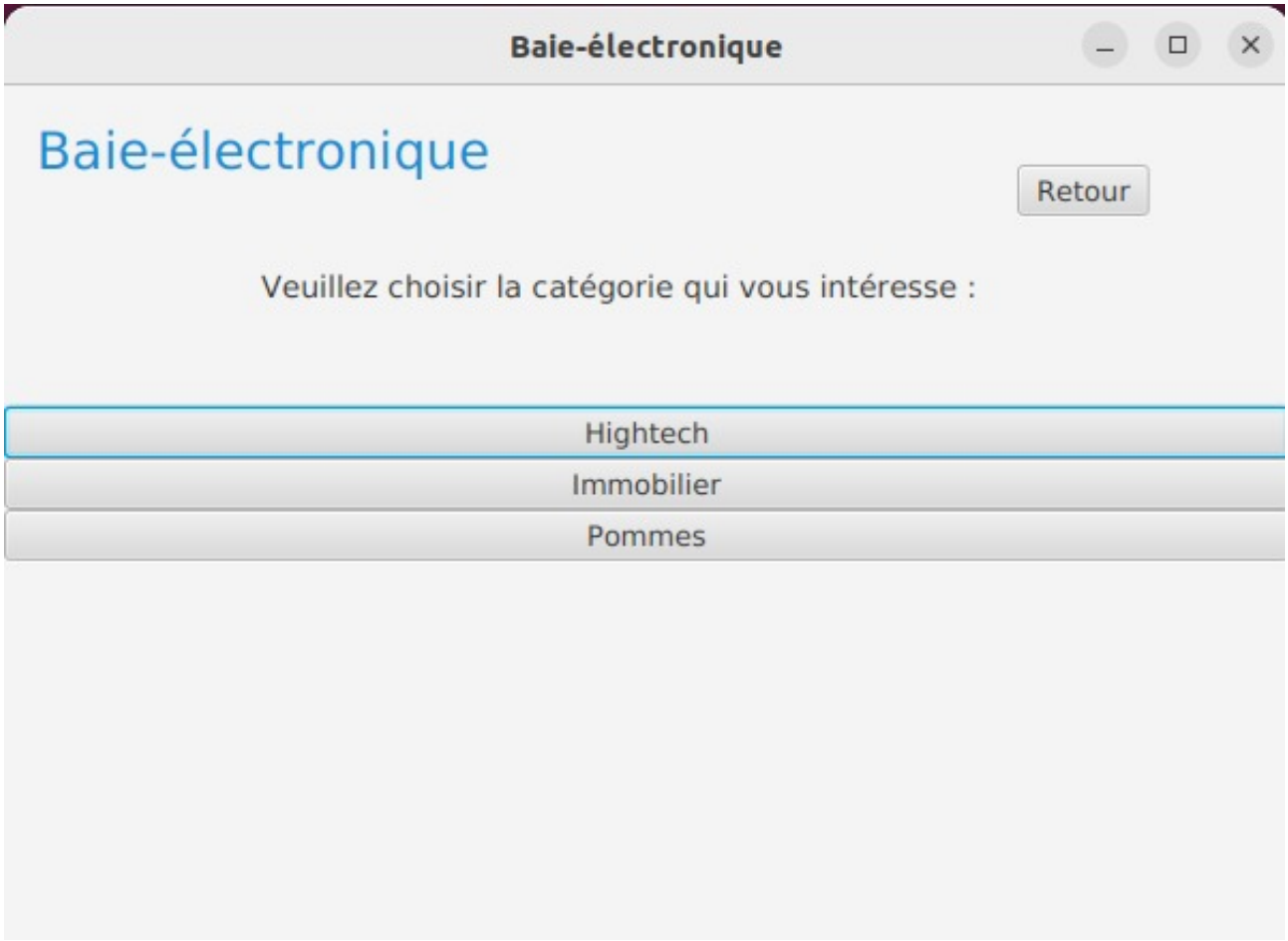

FIGURE 5 – Choix de la catégorie

FIGURE 6 – Choix de la salle de vente

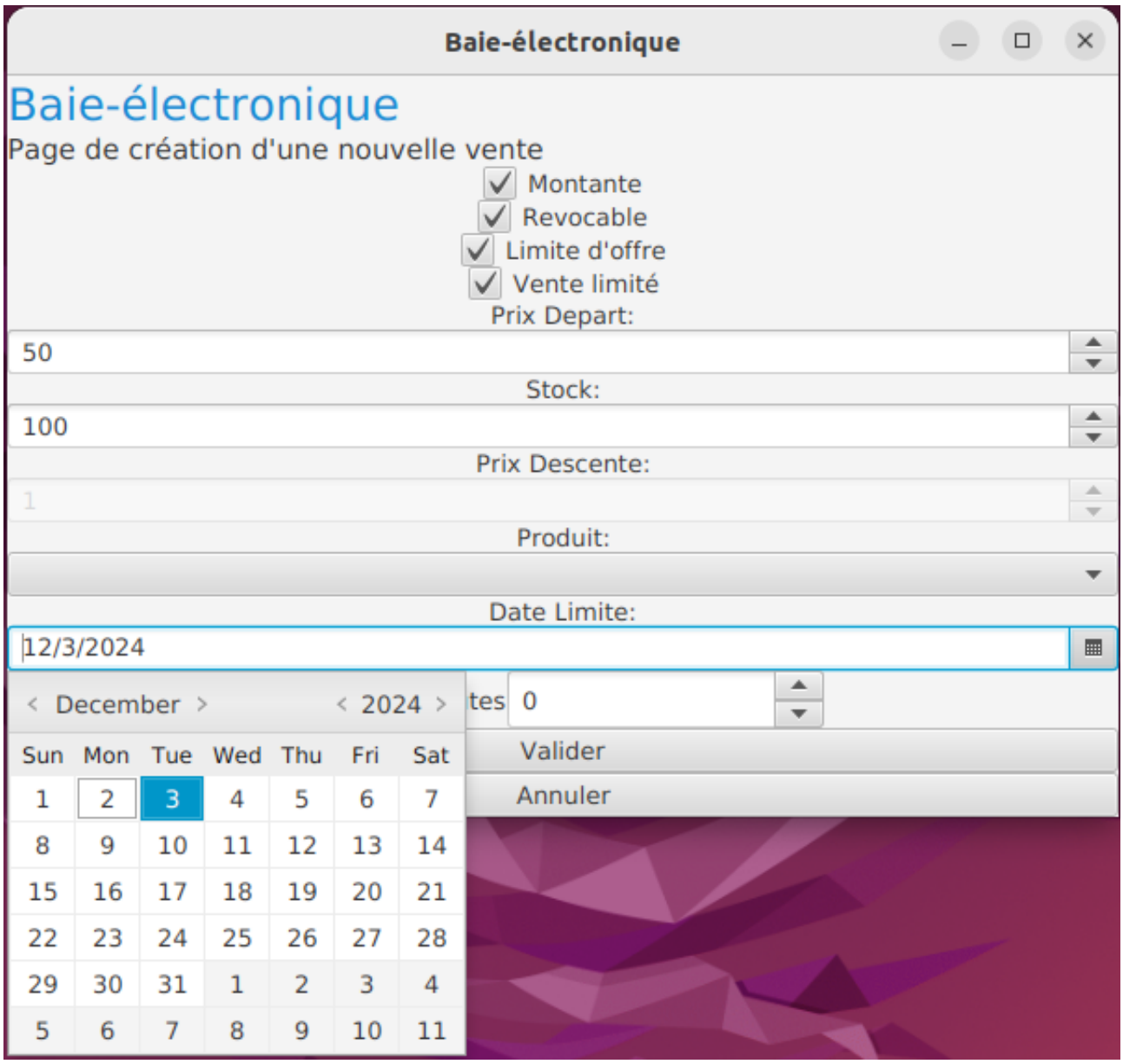

FIGURE 7 – Ajout de la vente