Cave à vins (Partie 06)

# TP06 du module 02 – Spring Data

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Compléter les entités pour obtenir toutes les associations du projet * Création de requêtes spécifiques |

|  |
| --- |
| **Durée estimée** |
| 2 à 3 heures |

Contexte

En vous appuyant sur l’ensemble des concepts abordés dans le cours, mettez en place une application Spring Boot permettant la gestion d’une cave à vins.

L’application de cave à vin sera une application d’API Web sécurisée à la fin des itérations :

* Elle permet de gérer le stock des bouteilles de vins par le propriétaire qui est un administrateur de l’application
* Elle permet de gérer l’achat en ligne des bouteilles par un client
  + Les bouteilles de vin peuvent avoir 3 couleurs : Rouge, Blanc ou Rosé
  + Les bouteilles de vin sont associées aux régions françaises
* Un client et un propriétaire doivent s’authentifier
* Un visiteur est un client qui n’a pas de compte
* Il pourra uniquement consulter le stock

Dans les itérations nous réaliserons uniquement le back de l’application.

Cette application se fera avec Spring Boot.

## Voici le diagramme des cas d’utilisation complet :

Une image contenant texte, diagramme

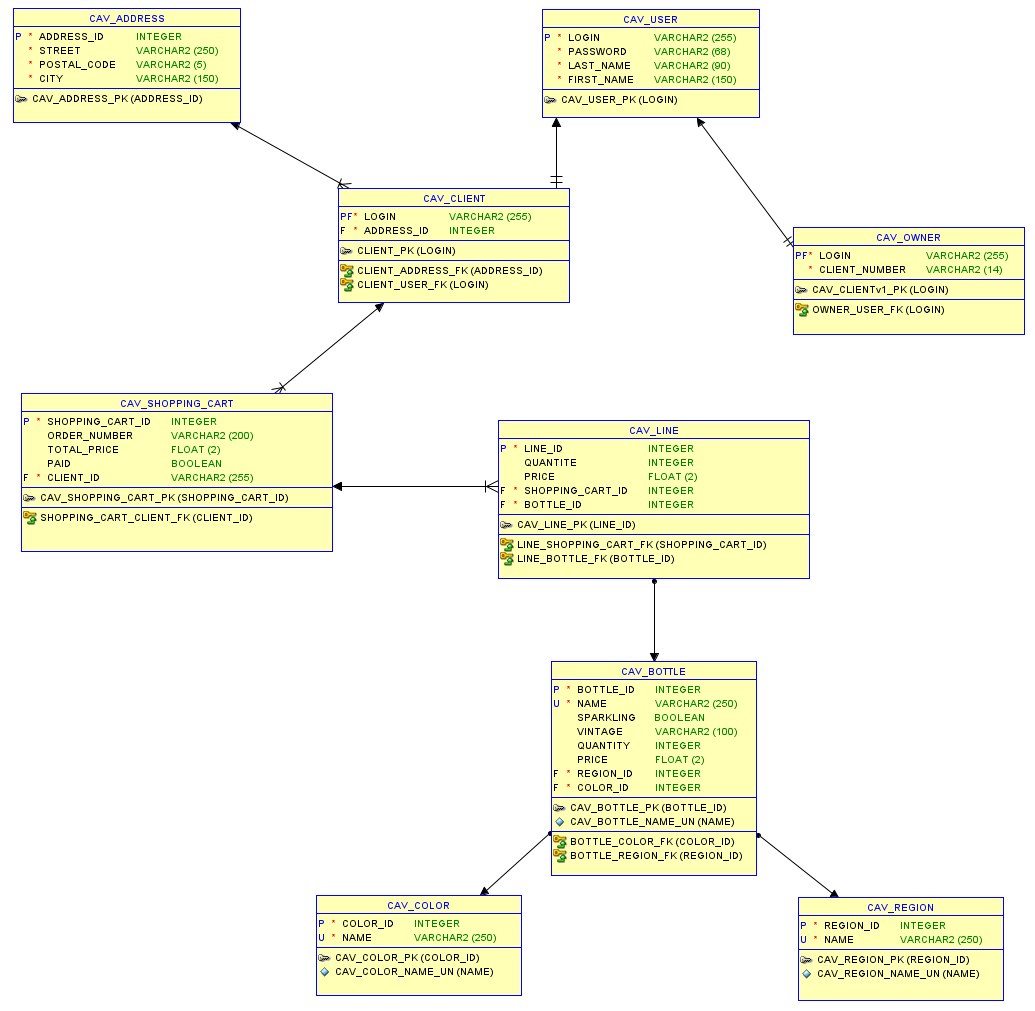
Description générée automatiquement

* Voici le diagramme complet des entités :

Une image contenant texte, diagramme, Parallèle, Plan

Description générée automatiquement

* Voici le diagramme des tables en base de données :



Énoncé

# Réalisation :

Les niveaux 1 sont obligatoires,

Les niveaux 2 sont plus poussés et optionnels

# Contexte :

Les couleurs, les régions et les bouteilles sont stockées par le propriétaire. Les paniers sont créés par les clients. Pour simuler la création, la mise à jour ou la suppression d’une ligne de panier, il faut avoir les 3 éléments en base.

## **Niveau 1** : Association ManyToOne LignePanier-Bouteille

1. Ajouter l’association ManyToOne entre LignePanier et Bouteille
2. Une classe de tests unitaires appelée TestManyToOneLignePanierBouteille vous est fournie

* Elle contient une méthode pour initialiser :
  + initDB annotée @BeforeEach, qui stocke en base
  + 1 couleur (Blanc)
  + 1 région
  + 1 bouteille
* En vous aidant des tests unitaires des démonstrations et des TP précédents
  + Créer 2 tests unitaires :
  + Test\_save 🡪 sauvegarde une ligne associée à sa bouteille
  + Test\_delete 🡪 valide la suppression de la ligne mais la conservation de la bouteille
* Voici le genre de traces à obtenir (les retours chariots et décalages sont là pour simplifier la lecture, ils ne font pas parti des traces d’origine)
  + Voici la ligne et sa bouteille :

LignePanier(

id=1,

quantite=4,

prix=95.8,

bouteille=Bouteille(id=1,

nom=DOMAINE ENI Ecole,

petillant=false,

millesime=2022,

quantite=1298, prix=23.95,

region=Region(id=1, nom=Pays de la Loire),

couleur=Couleur(id=1, nom=Blanc)))

## **Niveau 1** : Requêtes spécifiques

Dans notre application, il faut gérer des recherches spécifiques :

* Des bouteilles par une région
* Des bouteilles par une couleur
* Un utilisateur par son pseudo
* Un utilisateur par son pseudo et son mot de passe

1. Ces recherches doivent être des requêtes de Spring Data JPA avec mots clefs
2. Une classe de tests unitaires appelée TestRequetes vous est fournie

* Elle contient une méthode pour initialiser :
  + initDB annotée @BeforeEach, qui stocke en base
  + 3 Couleur (Rouge, Blanc, Rosé)
  + 3 Region (Grand Est, Pays de la Loire, Nouvelle-Aquitaine)
  + 5 Bouteille
  + 3 Utilisateur (1 Utilisateur quelconque, 1 Client et 1 Proprio)
* Créer des tests unitaires pour valider chaque requête

## **Niveau 2** : Association ManyToOne Panier-Client

1. Ajouter l’association ManyToOne entre Panier et Client
2. Une classe de tests unitaires appelée TestManyToOnePanierClient vous est fournie

* Elle contient :
  + Une méthode initDB annotée @BeforeEach, qui stocke en base
    - Stocker en base 2 couleur (Blanc, Rouge)
    - Stocker en base 2 régions (Pays de la Loire et Grand Est)
    - Stocker en base 2 bouteilles
  + Une méthode jeudDeDonnees pour créer 2 Panier
    - A vous de l’utiliser dans vos tests
* En vous aidant des tests unitaires des démonstrations et des TP précédents
* Créer 3 tests unitaires :
  + test\_save\_1panier 🡪 sauvegarde un Panier et son Client associé
  + test\_save\_paniers\_unClient 🡪 sauvegarder plusieurs Panier d’un même Client
  + test\_delete 🡪 montrer que supprimer le Client ne supprime pas les Panier
* Voici le genre de traces à obtenir (les retours chariots et décalages sont là pour simplifier la lecture, ils ne font pas parti des traces d’origine)
  + Voici pour un Panier et son Client :

Panier(id=1, numCommande=null, prixTotal=34.35, paye=false,

lignesPanier=[LignePanier(id=1, quantite=3, prix=34.35,

bouteille=Bouteille(id=1, nom=DOMAINE ENI Ecole,

petillant=false, millesime=2022,

quantite=1298, prix=11.45,

region=Region(id=1, nom=Pays de la Loire),

couleur=Couleur(id=1, nom=Blanc)))]

* + Voici pour 2 Panier et le même Client :

[ Panier(id=1, numCommande=null, prixTotal=34.35, paye=false,

lignesPanier=[LignePanier(id=1, quantite=3, prix=34.35,

bouteille=Bouteille(id=1, nom=DOMAINE ENI Ecole,

petillant=false, millesime=2022,

quantite=1298, prix=11.45,

region=Region(id=1, nom=Pays de la Loire),

couleur=Couleur(id=1, nom=Blanc)))],

client=Utilisateur(pseudo=bobeponge@email.fr, nom=Eponge, prenom=Bob)),

Panier(id=2, numCommande=null, prixTotal=239.5, paye=false,

lignesPanier=[LignePanier(id=2, quantite=10, prix=239.5,

bouteille=Bouteille(id=2, nom=DOMAINE ENI Service,

petillant=false, millesime=2015,

quantite=2998, prix=23.95,

region=Region(id=3, nom=Nouvelle-Aquitaine),

couleur=Couleur(id=2, nom=Rouge)))],

client=Utilisateur(pseudo=bobeponge@email.fr, nom=Eponge, prenom=Bob))]

## **Niveau 2** : Requêtes spécifiques plus poussées

Dans notre application, il faut ajouter des recherches plus spécifiques :

* Les paniers d’un client (les paniers qui ont un numéro de commande nul)
* Les commandes d’un client (les paniers qui ont un numéro de commande non nul)

1. Déterminer 2 solutions pour la première recherche :
   * 1 solution avec une requête JPQL
   * 1 solution avec une requête Spring Data JPA par mots clefs
2. Déterminer 2 solutions pour la seconde recherche :
   * 1 solution avec une requête native
   * 1 solution avec une requête Spring Data JPA par mots clefs
3. Une classe de tests unitaires appelée TestRequetesPlusPoussees vous est fournie

* Elle contient une méthode pour initialiser :
  + initDB annotée @BeforeEach, qui stocke en base
  + 3 Couleur (Rouge, Blanc, Rosé)
  + 3 Region (Grand Est, Pays de la Loire, Nouvelle-Aquitaine)
  + 5 Bouteille
  + 3 Utilisateur (1 Utilisateur quelconque, 1 Client et 1 Proprio)
  + Des LignePanier et 3 Panier associé à Tom le Client
  + Dont 2 sont payés et commandés
* Créer des tests unitaires pour valider chaque requête