Cave à vin (partie 03)

# TP03 du module 02 – Spring Data

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Création d’une association OneToMany unidirectionnelle |

|  |
| --- |
| **Durée estimée** |
| 1 heure |

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 vin.

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 des entités à réaliser :

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Voici le diagramme des tables en base de données :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Énoncé

# Créer les entités

* Le package des classes est fr.eni.cave.bo.client
* Utiliser les annotations de JPA
  + Appliquer les annotations pour l’association OneToMany unidirectionnelle
  + Penser à préciser l’annotation @JoinColumn
* Les prix doivent être des float avec 2 décimales
  + Utiliser pour cela le paramètre precision=2 de l’annotation @Column
  + JAVADOC : int precision à (Optional) The precision for a decimal (exact numeric) column
* Utiliser les annotations de Lombok
  + Eviter que la liste soit nulle ; risque de NullPointerException
    - Utiliser l’annotation @Builder.Default
  + Pour les 2 entités, seul l’attribut id est utilisé pour la méthode equals

# Créer les Repository

# Créer des tests unitaires

* Une classe de tests unitaires appelée TestOneToManyUni vous est fournie
* Elle contient une méthode pour stocker en base :
  + 1 Panier
  + 1 LignePanier
* Aidez-vous de cette méthode et des tests unitaires des démonstrations
* Créer les tests unitaires pour cette association
  + Créer des prix avec au moins 2 décimales pour bien montrer le comportement
  + L’application peut créer un nouveau panier à partir de sa première ligne
    - Le panier a un identifiant à nul et est vide
    - Seule la ligne a des informations
  + Il est possible d’ajouter une ligne à un panier existant
* Il faut 4 tests :
  + test\_save\_nouvelleLigne\_nouveauPanier
  + test\_save\_nouvelleLigne\_Panier()
  + test\_delete()
  + test\_orphanRemoval()
* Voici le genre de traces à obtenir :
  + Le panier créé par le test test\_save\_nouvelleLigne\_nouveauPanier

Panier(id=1, numCommande=null, prixTotal=95.8, paye=false, lignesPanier=[LignePanier(id=1, quantite=4, prix=95.8)])

* + Le panier mis à jour par le test test\_save\_nouvelleLigne\_Panier
    - Initialement :

Panier(id=2, numCommande=null, prixTotal=34.35, paye=false, lignesPanier=[LignePanier(id=2, quantite=3, prix=34.35)])

* + - Après l’ajout de la nouvelle ligne :

Panier(id=2, numCommande=null, prixTotal=273.85, paye=false, lignesPanier=[LignePanier(id=2, quantite=3, prix=34.35), LignePanier(id=3, quantite=10, prix=239.5)])