Cave à vin (partie 10)

# TP04 du module 04 – Spring Web Service

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Ajout du starter de validation * Mise en place des annotations de validation sur le BO * Activation de la validation sur le contrôleur |

|  |
| --- |
| **Durée estimée** |
| 30 minutes |

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.

Dans les itérations de ce module, nous allons développer l’API Web.

Solution

# Réalisation :

Les niveaux 1 sont obligatoires,

Les niveaux 2 sont plus poussés et optionnels

# Niveau 1 :

## Ajouter le starter spring-boot-starter-validation

* Dans build.gradle :

//Validation des BO

**implementation** 'org.springframework.boot:spring-boot-starter-validation'

* + Refresh Gradle Project

## Sur Bouteille ; ajouter les contraintes de validation

* + nom : @NotBlank et @Size
  + millesime : @Size
  + quantite et prix : @Min à 1
  + region : @NotNull
  + couleur : @NotNull

**package** fr.eni.cave.bo.vin;

**import** jakarta.persistence.\*;

**import** jakarta.validation.constraints.\*;

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@EqualsAndHashCode

@ToString

@Builder

@Entity

@Table(name = "CAV\_BOTTLE")

**public** **class** Bouteille {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "BOTTLE\_ID")

**private** Integer id;

**@NotBlank**

**@Size(max = 250)**

@Column(name = "NAME", length = 250, unique = **true**, nullable = **false**)

**private** String nom;

@Column(name = "SPARKLING")

**private** **boolean** petillant;

**@Size(max = 100)**

@Column(name = "VINTAGE", length = 100)

**private** String millesime;

**@Min(1)**

@Column(name = "QUANTITY")

**private** **int** quantite;

**@Min(1)**

@Column(name = "PRICE", precision = 2)

**private** **float** prix;

**@NotNull**

@EqualsAndHashCode.Exclude

@ManyToOne

@JoinColumn(name = "REGION\_ID")

**private** Region region;

**@NotNull**

@EqualsAndHashCode.Exclude

@ManyToOne

@JoinColumn(name = "COLOR\_ID")

**private** Couleur couleur;

}

## Activer la validation sur la méthode POST

**…**

**import** jakarta.validation.Valid;

…

@PostMapping

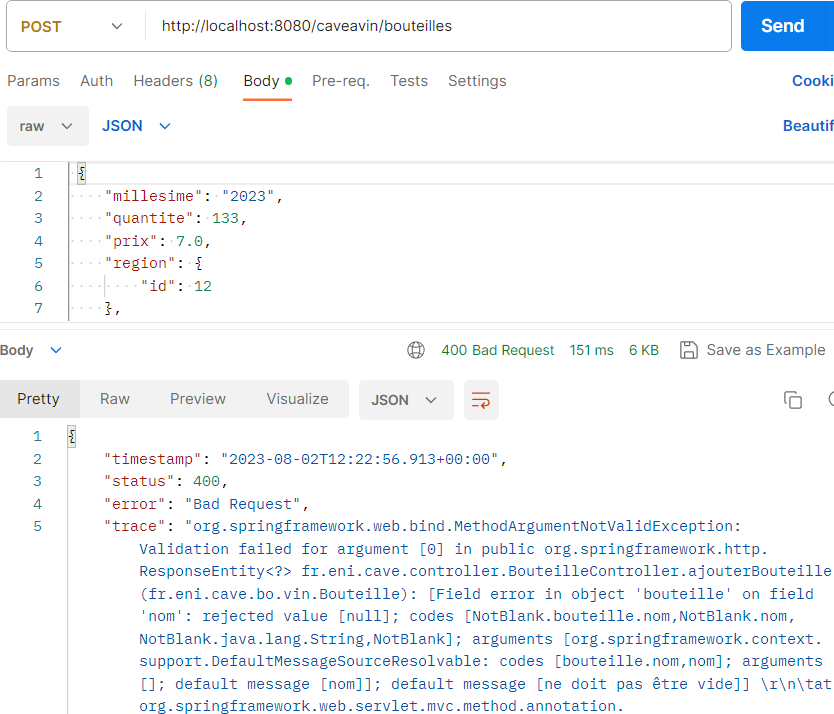
**public** ResponseEntity<?> ajouterBouteille**(@Valid** @RequestBody Bouteille bouteille) {

**…**

}

## Tester avec Postman

* La validation de chaque cas
  + Valider @NotBlank sur le nom :
    - Appeler le POST sans le champ nom



* + - Le statut est 400
    - Et nous avons la trace d’exception de MethodArgumentNotValidException
  + Tous les autres cas sont similaires

# Niveau 2 :

## Activer la validation sur la méthode PUT

@PutMapping

**public** ResponseEntity<?> miseAJourBouteille(**@Valid** @RequestBody Bouteille bouteille) {

…

}

## Tester avec Postman

* La validation de chaque cas
  + Et identique à la version de niveau 1
  + La seule différence est qu’on teste la méthode PUT