Cave à vins (Partie 07)

# TP01 du module 04 – Spring Web Service

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Configurer le projet avec le starter Spring Web * Créer un contrôleur REST * Créer une méthode GET |

|  |
| --- |
| **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 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.

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

* Si vous n’aviez pas fini le TP du module 02. Vous trouverez les classes solutions dans un zip des ressources.

## 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

Solution

# Copier la couche BLL fournie

Pour faire gagner du temps, et vous concentrer sur les contrôleurs ; nous vous fournissons une partie de la couche BLL

* Vous trouverez l’interface BouteilleService et son implémentation dans les ressources
* Copier les dans fr.eni.cave.bll
* Voici le diagramme des classes :

Une image contenant texte, capture d’écran, Police, noir et blanc

Description générée automatiquement

* Les méthodes charger sont pour les profils visiteur, Client et Proprio
* Les méthodes ajouter (= ajouter ou modifier) ou supprimer sont pour le Proprio

## Modifier BouteilleRespository

* Pour faire fonctionner le code, il faut lui ajouter la méthode :

// Rechercher une bouteille par son nom

Bouteille findByNom(String nom);

# Configurer le projet en projet Web

* Ajout du starter Spring Web
  + Ouvrir build.gradle :
  + Ajout du starter Spring Web

//Ajout du starter Spring Web

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

* Puis, clic droit sur le projet
  + Gradle 🡪 Refresh Gradle Project
* Le projet est devenu un projet Web
  + Vous pouvez voir dans les dépendances
    - spring-boot-starter-web-3.X.X.jar
    - spring-web-6.X.X.jar
    - jackson-…jar
    - …
  + Ces librairies permettent d’accéder au contexte Web et REST avec Spring Boot

# Mettre le mode update pour la base

* Dans le fichier application.properties
  + Modifier create en update

spring.jpa.hibernate.ddl-auto=create

# Créer le contrôleur BouteilleController

* L’URL racine est : /caveavin/bouteilles

@RestController

@RequestMapping("/caveavin/bouteilles")

**public** **class** BouteilleController {

* Injecter lui le service
  + Utilisation de l’annotation Lombok @AllArgsConstructor pour injecter par constructeur

**private** BouteilleService bService;

* Définissez une méthode Get pour retourner l’ensemble des bouteilles
  + Si la liste est vide retourner un statut 204
  + Pour gérer un statut spécifique et des informations différentes, il faut utiliser ResponseEntity< ?>

**package** fr.eni.cave.controller;

**import** java.util.List;

**import** org.springframework.http.\*;

**import** org.springframework.web.bind.annotation.\*;

**import** fr.eni.cave.bll.BouteilleService;

**import** fr.eni.cave.bo.vin.Bouteille;

**import** lombok.AllArgsConstructor;

@AllArgsConstructor

@RestController

@RequestMapping("/caveavin/bouteilles")

**public** **class** BouteilleController {

**private** BouteilleService bService;

// Pour visiteur, Client et Proprio

@GetMapping

**public** ResponseEntity<?> rechercherTousBouteilles() {

**final** List<Bouteille> bouteilles = bService.chargerToutesBouteilles();

**if** (bouteilles == **null** || bouteilles.isEmpty()) {

// Statut 204 : No Content - Pas de body car rien à afficher

**return** ResponseEntity.*noContent*().build();

}

// Statut 200 : OK + dans le body bouteilles

// Le contenu du body est directement injecté dans la méthode ok

**return** ResponseEntity.*ok*(bouteilles);

}

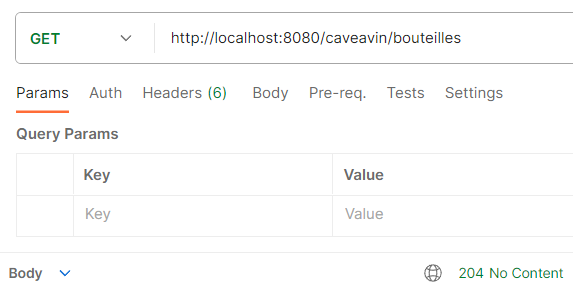
}

# Tester avec Postman

* Lancer l’application
* Votre base de données existe et les tables aussi

## Test sans donnée en base

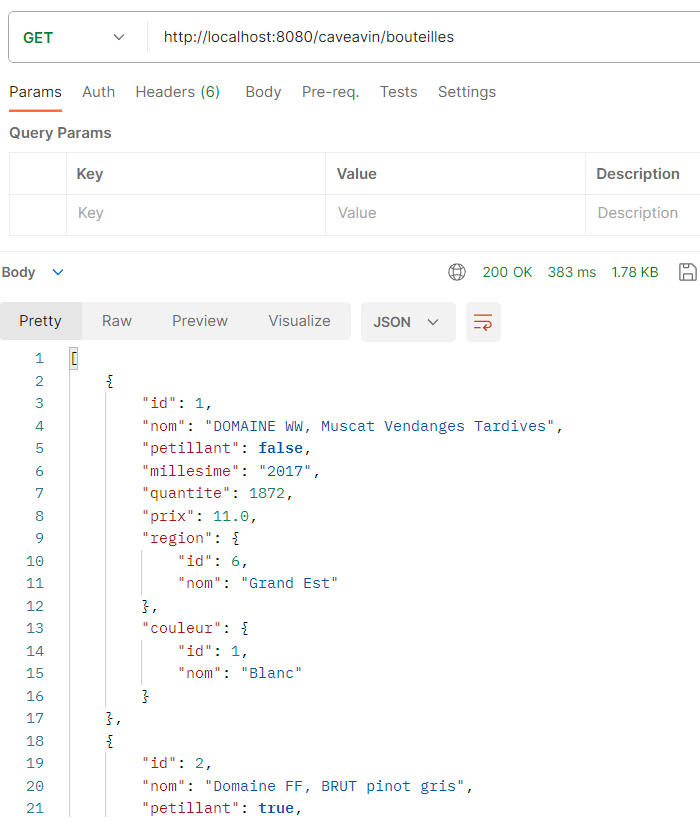
* + Faire un premier test, il n’y a pas de données en base donc un statut 204
* Création d’une requête GET sur l’URL : http://localhost:8080/caveavin/bouteilles



* Le statut est bien 204 🡪 pas de contenu à afficher

## Test avec des données en base

* Utiliser le fichier de script SQL (script\_bouteilles.sql) fourni pour mettre des données en base
* Valider le comportement de votre contrôleur
* Réexécuter la requête GET sur l’URL : http://localhost:8080/caveavin/bouteilles



* Le statut est 200
* Et nous avons bien le tableau des bouteilles en JSON qui est retourné