Cave à vins (Partie 12)

# TP01 du module 06 – Sécurité

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Ajout de Spring Security * Classe de configuration * Manipulation au travers de Postman |

|  |
| --- |
| **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 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 sécuriser notre API Web.

* Si vous n’aviez pas fini le TP du module 04. Vous trouverez les classes solutions dans un zip des ressources.
* Pensez à vérifier la configuration de Spring Boot dans build.gradle. Il faut ajouter toutes les librairies et starters du module 04

Solution

# Réalisation :

Les niveaux 1 sont obligatoires,

Les niveaux 2 sont plus poussés et optionnels

# Niveau 1 : Dépendances Spring Boot :

* Dans build.gradle ; ajouter le starter de sécurité

//Security

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

**testImplementation** 'org.springframework.security:spring-security-test'

* Faire un Gradle Refresh Project
* Votre projet est à présent un projet sécuriser par Spring Security

# Niveau 1 : Gestion des utilisateurs et leurs rôles :

Nous avons déjà une hiérarchie d’entités Utilisateur, Proprio et Client. L’entité Utilisateur représente les données d’un utilisateur à connecter :

* Un pseudo
* Un mot de passe

Il manque cependant la notion de rôle. Actuellement cette notion est portée par les classes d’héritages : Proprio et Client

* Pour manipuler facilement la sécurité et la notion de rôles.
* Nous allons ajouter une colonne à la table « cav\_user » pour gérer les rôles

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

Description générée automatiquement

## Mise à jour de l’entité Utilisateur

* Mettre à jour l’entité Utilisateur pour correspondre à cette table
* Ajouter le nouvel attribut « authority », préciser la taille de la colonne.

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

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

**import** lombok.\*;

**import** lombok.experimental.SuperBuilder;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@EqualsAndHashCode(of = { "pseudo" })

@ToString(of = { "pseudo", "nom", "prenom" })

@SuperBuilder

@Entity

@Table(name = "CAV\_USER")

@Inheritance(strategy = InheritanceType.***JOINED***)

**public** **class** Utilisateur {

@Id

@Column(name = "LOGIN", nullable = **false**, length = 255)

**private** String pseudo;

@Column(name = "PASSWORD", nullable = **false**, length = 68)

**private** String password;

@Column(name = "LAST\_NAME", nullable = **false**, length = 90)

**private** String nom;

@Column(name = "FIRST\_NAME", nullable = **false**, length = 150)

**private** String prenom;

**@Column(length = 15)**

**private String authority;**

}

## Mise à jour de la base de données

Il faut supprimer nos données et nos tables. Pour avoir une infrastructure correcte et des données existantes

2 scripts vous sont fournis : drop\_table.sql et insert\_cave\_a\_vin.sql

* Exécuter le script drop\_table.sql
* Lancer votre application pour qu’elle crée l’ensemble des tables
* Exécuter le script insert\_cave\_a\_vin.sql
  + Il crée la table cav\_authorities
  + Et insère des données dans chaque table

Vous avez maintenant un ensemble de données en base.

Remarque dans le script :

* Le mot de passe est sur 68 caractères pour préciser l’algorithme de chiffrement au début de la chaîne {bcrypt}
* Les rôles sont préfixés par ROLE\_ pour suivre la configuration de Spring Security

# Niveau 1 : Configuration de sécurité

* Création de la classe de configuration
  + Création d’une méthode userDetailsManager
    - Elle prend en paramètre la DataSource
    - Elle retourne UserDetailsManager
    - Elle permet de créer les 2 requêtes, pour récupérer
      * L’identifiant, le mot de passe et l’activation d’un utilisateur
      * La liste des rôles de l’utilisateur
* Création de la classe AppConfigSecurity

**package** fr.eni.cave.security;

**import** javax.sql.DataSource;

**import** org.apache.commons.logging.\*;

**import** org.springframework.context.annotation.\*;

**import** org.springframework.security.provisioning.\*;

@Configuration

**public** **class** AppConfigSecurity {

**protected** **final** Log logger = LogFactory.*getLog*(getClass());

/\*\*

\* Récupération des utilisateurs de l'application via la base de données

\*/

@Bean

UserDetailsManager userDetailsManager(DataSource dataSource) {

JdbcUserDetailsManager jdbcUserDetailsManager = **new** JdbcUserDetailsManager(dataSource);

jdbcUserDetailsManager.setUsersByUsernameQuery("select login, password, 1 from cav\_user where login=?");

jdbcUserDetailsManager.setAuthoritiesByUsernameQuery("select login, authority from cav\_user where login=?");

**return** jdbcUserDetailsManager;

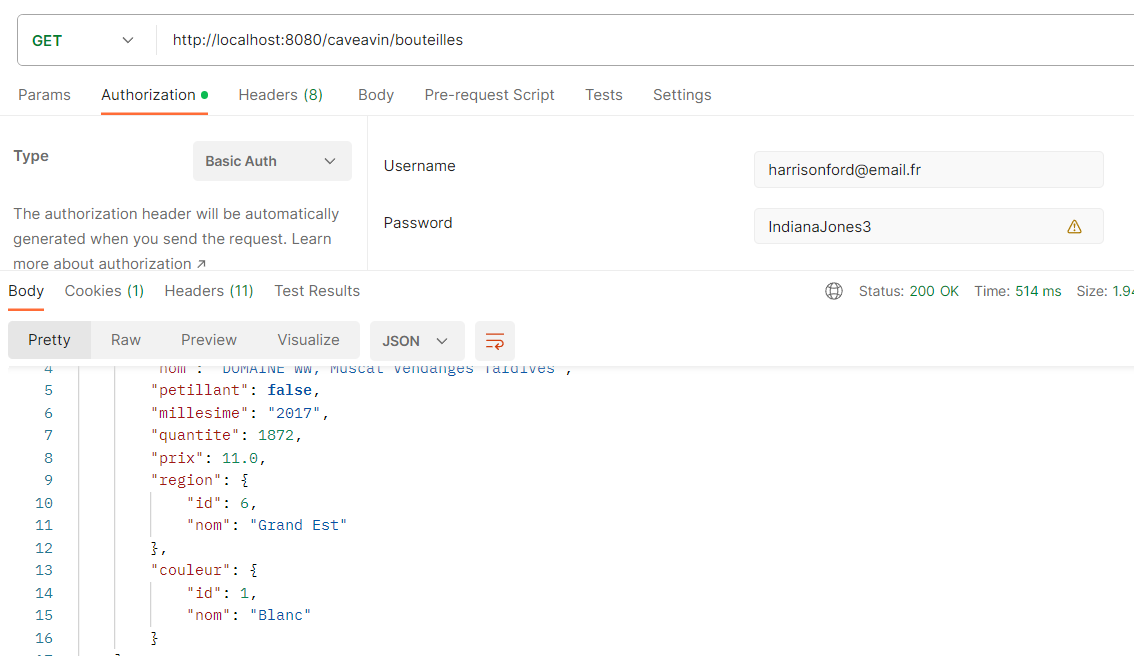
}

}

## Exécution dans Postman

Pour le moment, il faudra se connecter par défaut pour accéder à toutes URLs de l’application web services

* Création d’une nouvelle requête
  + Aujouter Authorization
  + Username : un des utilisateurs en base
  + Password : le mot de passe associé (il est en commentaire dans le script SQL)



# Niveau 2 : Accès limités des URLS selon les rôles

Pour la sécurité, nous voulons limiter l’accès aux ressources selon les rôles

## Contexte :

* Dans les ressources, vous avez 3 classes de fournies
  + PanierService et PanierServiceImpl 🡪 BLL pour gérer les Panier
  + PanierController 🡪 contrôleur REST pour Panier
* Copier les dans votre projet pour avoir plus de contraintes sur les URLs à gérer

## Limiter l’accès selon les rôles :

* Un client et un propriétaire doivent s’authentifier
  + Un client et un propriétaire peuvent
    - Consulter le stock des bouteilles 🡪 rechercher toutes les bouteilles
    - Rechercher un panier par son identifiant
    - Rechercher les paniers non payés d’un client
    - Rechercher les commandes d’un client (panier commandé et payé)
  + Un Client peut
    - Ajouter et mettre à jour un panier
    - Passer une commande (valider un panier et le payer)
  + Le propriétaire peut
    - Ajouter des bouteilles, des régions ou des couleurs
    - Modifier la quantité des bouteilles
    - Il peut consulter les régions et couleurs
* Un visiteur est un client qui n’a pas de compte
  + Il pourra uniquement consulter le stock des bouteilles 🡪 rechercher toutes les bouteilles
* Interdire tout autre URL

Modifier la classe de configuration de sécurité pour prendre en considération ces contraintes de rôles.

* Ajout de l’annotation pour gérer les filtres sur les URLs et les méthodes du protocole http :
  + @EnableWebSecurity
* Déclaration d’un bean de type SecurityFilterChain au travers d’une méthode filterChain ;
  + Elle prend en argument HttpSecurity
  + Permet de configurer les méthodes et url du protocole http accessible selon un certain rôle
  + Les ressources accessibles par tous
  + Déclaration d’une connexion via le formulaire par défaut

**package** fr.eni.cave.security;

**import** javax.sql.DataSource;

**import** org.apache.commons.logging.\*;

**import** org.springframework.context.annotation.\*;

**import** org.springframework.http.HttpMethod;

**import** org.springframework.security.config.Customizer;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.provisioning.\*;

**import** org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

**public** **class** AppConfigSecurity {

**protected** **final** Log logger = LogFactory.*getLog*(getClass());

/\*\*

\* Récupération des utilisateurs de l'application via la base de données

\*/

@Bean

UserDetailsManager userDetailsManager(DataSource dataSource) {

JdbcUserDetailsManager jdbcUserDetailsManager = **new** JdbcUserDetailsManager(dataSource);

jdbcUserDetailsManager.setUsersByUsernameQuery("select login, password, 1 from cav\_user where login=?");

jdbcUserDetailsManager.setAuthoritiesByUsernameQuery("select login, authority from cav\_user where login=?");

**return** jdbcUserDetailsManager;

}

/\*\*

\* Restriction des URLs selon la connexion utilisateur et leurs rôles

\*/

@Bean

SecurityFilterChain filterChain(HttpSecurity http) **throws** Exception {

http.authorizeHttpRequests(auth -> {

auth

// Consulter le stock des bouteilles - est accessible à tout le monde

.requestMatchers("/caveavin/bouteilles").permitAll()

// Permettre aux rôles CLIENT et OWNER de manipuler :

.requestMatchers(HttpMethod.***GET***, "/caveavin/paniers/\*\*").hasAnyRole("CLIENT", "OWNER")

.requestMatchers(HttpMethod.***GET***, "/caveavin/paniers/client/actifs/\*\*").hasAnyRole("CLIENT", "OWNER")

.requestMatchers(HttpMethod.***GET***, "/caveavin/paniers/client/commandes/\*\*").hasAnyRole("CLIENT", "OWNER")

// Restreindre au rôle CLIENT

.requestMatchers(HttpMethod.***POST***, "/caveavin/paniers").hasRole("CLIENT")

.requestMatchers(HttpMethod.***PUT***, "/caveavin/paniers").hasRole("CLIENT")

// Restreindre au rôle OWNER

.requestMatchers(HttpMethod.***GET***, "/caveavin/bouteilles/\*\*").hasRole("OWNER")

.requestMatchers(HttpMethod.***POST***, "/caveavin/bouteilles").hasRole("OWNER")

.requestMatchers(HttpMethod.***PUT***, "/caveavin/bouteilles").hasRole("OWNER")

.requestMatchers(HttpMethod.***DELETE***, "/caveavin/bouteilles").hasRole("OWNER")

// Toutes autres url et méthodes HTTP ne sont pas permises

.anyRequest().denyAll();

});

// Use Http Basic Authentication

http.httpBasic(Customizer.*withDefaults*());

// Désactivé Cross Site Request Forgery

// Non préconisé pour les API REST en stateless.

// Sauf pour POST, PUT, PATCH et DELETE

http.csrf(csrf -> {

csrf.disable();

});

**return** http.build();

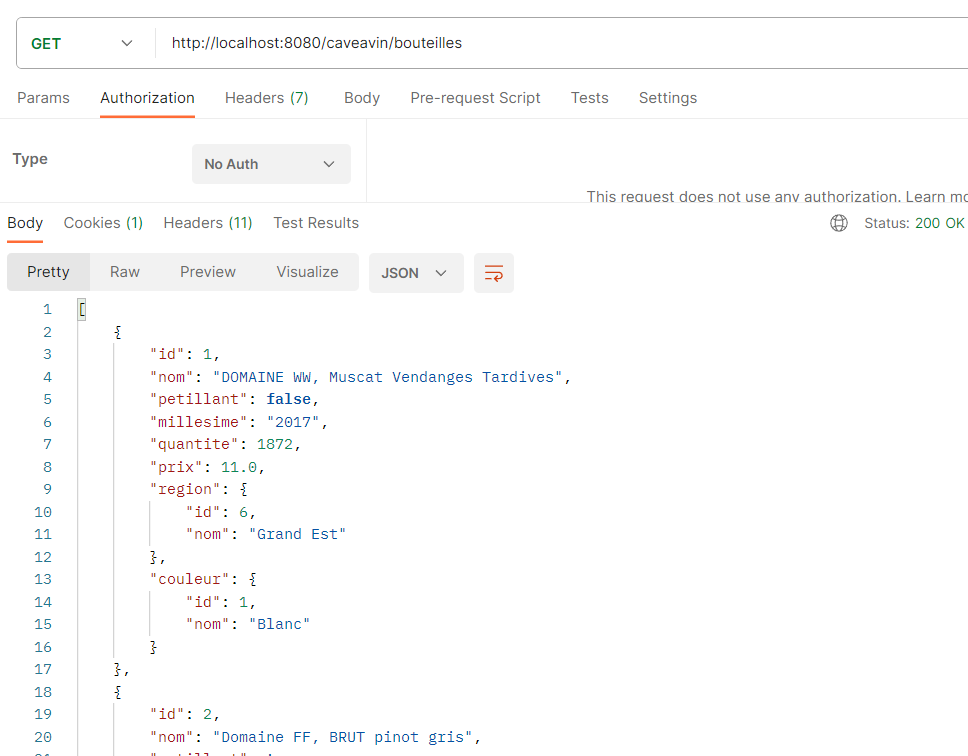
}

}

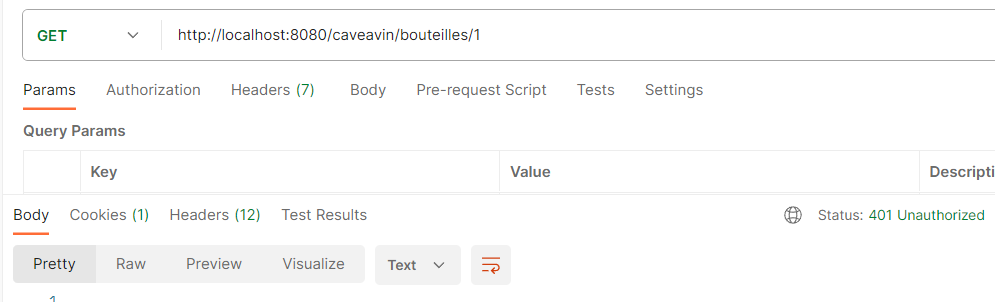
* Postman est un client http indépendant de notre application et du navigateur
  + Ajout de CSRF en disable
    - Pour permettre de manipuler l’API REST au travers d’outil autre que le navigateur web (ex : Postman)
  + **CSRF** (Cross-Site Request Forgery) est une attaque qui usurpe l'identité d'un utilisateur de confiance et envoie des commandes non désirées sur un site web.
    - Cela peut être réalisé, par exemple, en ajoutant des paramètres malveillants dans un URL associé à un lien qui prétend aller quelque part ailleurs.
  + Spring Security préconise de ne pas le désactiver
    - Mais pour nos manipulations au travers de Postman, nous le faisons
    - Bien se rendre compte que c’est nécessaire pour les POST, PUT, PATCH et DELETE
    - Pour les GET nous pouvions laisser par défaut le CSRF actif.

## Exécution

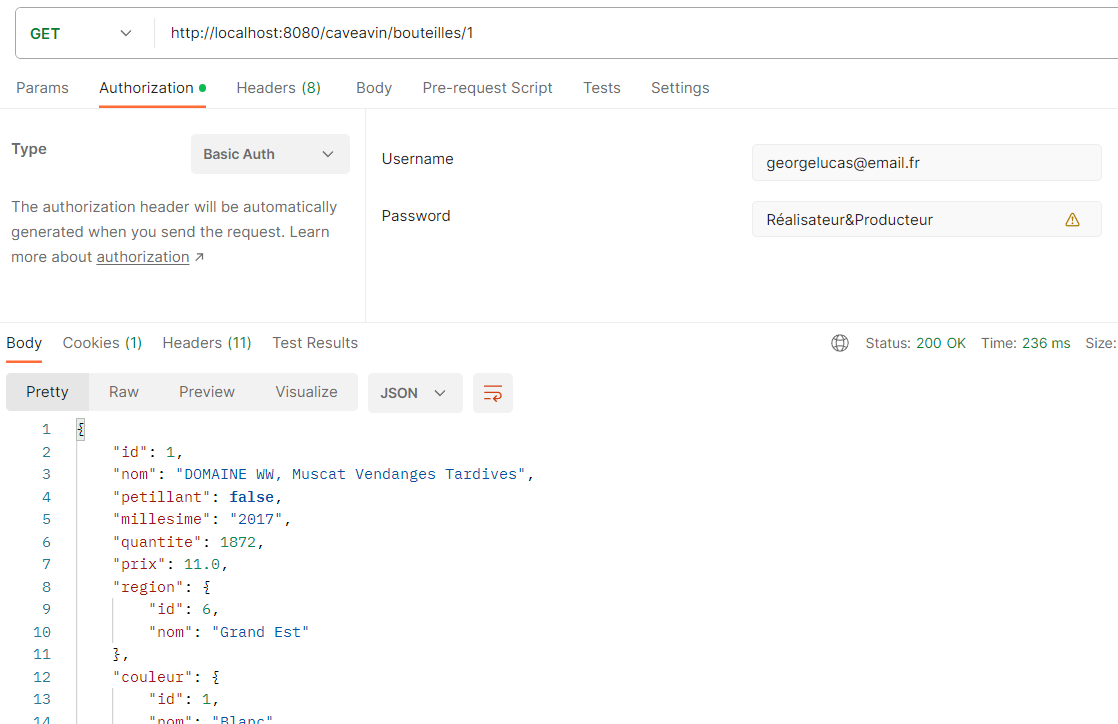
* Tout le monde peut accéder au stock des bouteilles :



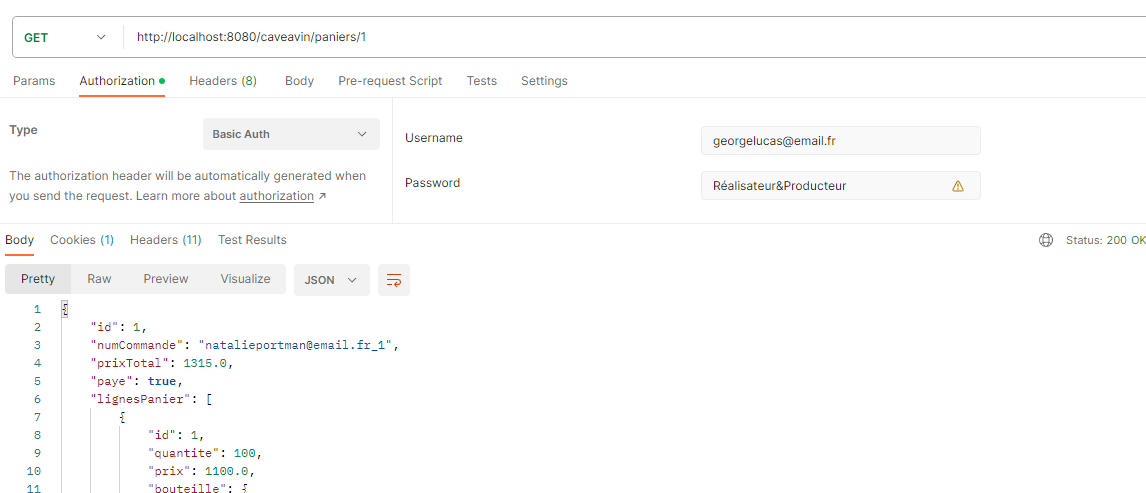
* Dès qu’il faut accéder à un autre URL ; il y a un 401 car il faut s’authentifier



* Pour toutes autres ressources associées au contrôleur BouteilleController, il faut le rôle OWNER pour être authorisé
  + Exemple : rechercherBouteilleParId
  + Utiliser : georgelucas@email.fr / Réalisateur&Producteur



* + Si vous tester, avec un utilisateur qui a un compte CLIENT, le statut est 403
  + Si vous tester sans utilisateur, le statut est 401
* Pour PanierController ; le rôle OWNER a droit sur les méthodes GET déclarées
  + Exemple : Récupérer le Panier d’identifiant 1



* + Pour POST et PUT, il n’est pas autorisé 🡪 statut 403
* Pour PanierController : le rôle CLIENT a droit à tout
  + Exemple : Créer un nouveau panier
    - Voici le body :

{

        "prixTotal": 215.0,

        "paye": **false**,

        "lignesPanier": [

            {

                "quantite": 10,

                "prix": 215.0,

                "bouteille": {

                    "id": 2,

                    "nom": "Domaine FF, BRUT pinot gris",

                    "petillant": **true**,

                    "millesime": "2020",

                    "quantite": 972,

                    "prix": 21.5,

                    "region": {

                        "id": 6,

                        "nom": "Grand Est"

                    },

                    "couleur": {

                        "id": 1,

                        "nom": "Blanc"

                    }

                }

            }

        ],

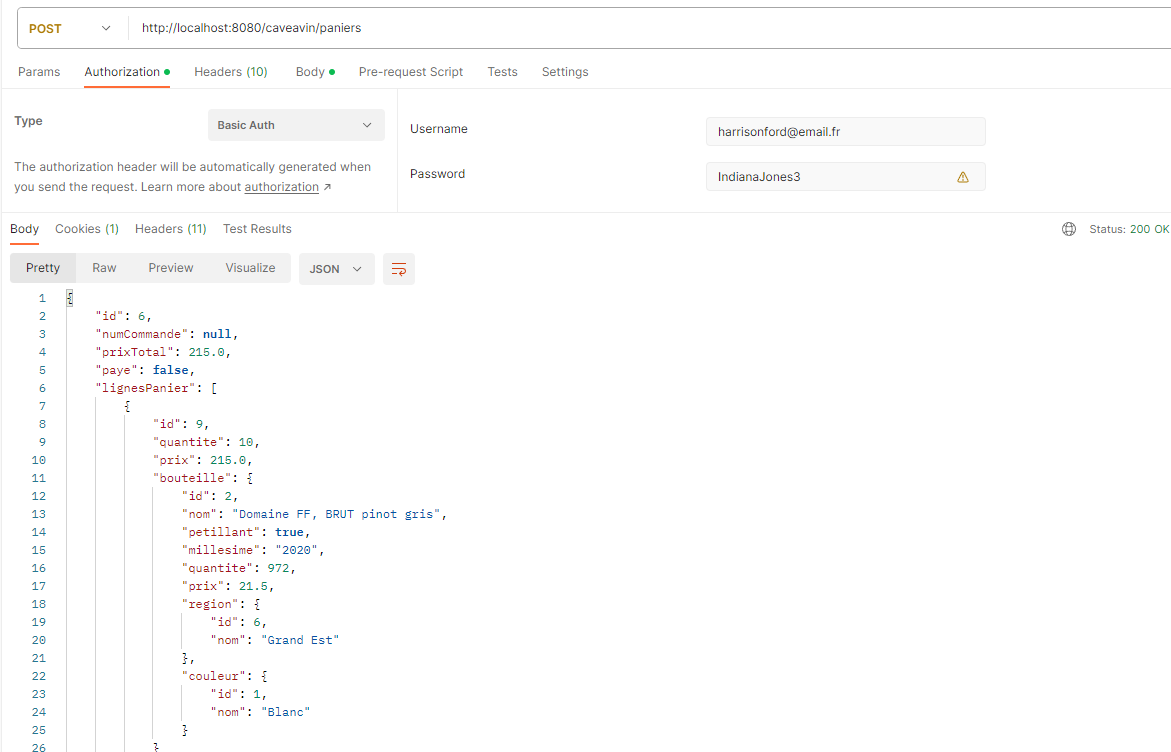
        "client": {

            "pseudo": "harrisonford@email.fr"

        }

    }

* + - Et le Client : harrisonford@email.fr / IndianaJones3
  + Il y a création d’un nouveau Panier en base :



* Si on utilise de nouveau ce Panier, pour valider la commande avec PUT
  + Le Panier est mis à jour
  + Le numéro de commande est créé et l’attribut paye est passé à vrai

{

    "id": 6,

    "numCommande": "harrisonford@email.fr\_6",

    "prixTotal": 215.0,

    "paye": **true**,

    "lignesPanier": [

        {

            "id": 9,

            "quantite": 100,

            "prix": 215.0,

            "bouteille": {

                "id": 2,

                "nom": "Domaine FF, BRUT pinot gris",

                "petillant": **true**,

                "millesime": "2020",

                "quantite": 972,

                "prix": 21.5,

                "region": {

                    "id": 6,

                    "nom": "Grand Est"

                },

                "couleur": {

                    "id": 1,

                    "nom": "Blanc"

                }

            }

        }

    ],

    "client": {

        "pseudo": "harrisonford@email.fr",

        "password": "{bcrypt}$2a$10$/5nEPZg4/BC/Jj0Q/yzEQ.0gGWzBu0ZmziWaeIAVtrtVj.VoAHPHS",

        "nom": "Ford",

        "prenom": "Harrison",

        "adresse": {

            "id": 3,

            "rue": "Rue du Temple",

            "codePostal": "35000",

            "ville": "Rennes"

        }

    }

}