Cave à vin (partie 01)

# TP01 du module 02 – Spring Data

|  |
| --- |
| Ce TP sera un TP fil rouge du cours   * Mise en place de Spring Boot et Spring Core * Création des premières entités JPA * Création des premiers Repository de Spring Data JPA * Utilisation de tests unitaires pour valider |

|  |
| --- |
| **Durée estimée** |
| 30 minutes et 1 heure |

Solution

# Créer une application CaveAVin avec SpringBoot

* Utiliser Spring initializr et sélectionner :
  + Gradle (privilégier avec le langage Groovy)
  + Java en version minimum 17
  + La version courante de Spring Boot (Release)
  + Le packaging est en Jar

## Métadonnées :

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

Description générée automatiquement

## Dépendances Spring Boot :

* Lombok
* Spring Boot DevTools
* Spring Data JPA
* MS SQL Server Driver
* Importer le projet dans votre IDE
* Ajouter la dépendance vers la base de tests H2 dans build.gradle :

//Ajout de la dépendance de la datasource H2 pour les tests unitaires sur JPA

testImplementation 'com.h2database:h2'

* Clic droit sur le projet
  + Gradle à Refresh Gradle Project

# Configurer la base de données

* Créer une base de données appelée : CAVE\_A\_VIN

## Configurer la source de données associées

* Pour le moment, nous allons surtout manipuler notre couche d’accès aux données au travers des tests unitaires et donc de la base de données embarquées H2
* Mais en prévision de la mise en place de la couche métier et des API, nous allons configurer dès maintenant l’utilisation de la base de données relationnelles Microsoft SQL Server
* Il est possible de la configurer la source de données via :
  + Le fichier application.properties qui est le choix par défaut
  + Ou via le fichier application.yml avec la grammaire correspondante
  + Pour cette itération, nous vous fournissons les 2 choix

## Déclaration dans application.properties

#Connexion à la base de données

spring.datasource.url=jdbc:sqlserver://localhost;databasename=CAVE\_A\_VIN;integratedSecurity=false;encrypt=false;trustServerCertificate=false

spring.datasource.username=sa

spring.datasource.password=Pa$$w0rd

#Détail des requêtes pour le développement

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

## Déclaration dans application.yml

#Connection to DB

spring:

datasource:

url: jdbc:sqlserver://localhost;databasename=CAVE\_A\_VIN;integratedSecurity=false;encrypt=false;trustServerCertificate=false

username: sa

password: Pa$$w0rd

#Options to DB

jpa:

show-sql: true

hibernate:

ddl-auto: create

# Créer l’entité Client

## Voici le diagramme de l’entité à réaliser :

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

Description générée automatiquement

Voici le diagramme de la table en base de données :

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

Description générée automatiquement

* Le package de la classe est fr.eni.cave.bo.client
* Utiliser les annotations de JPA
* Utiliser les annotations de Lombok
  + Utiliser uniquement l’attribut pseudo pour la méthode equals
    - Sur l’annotation @EqualsAndHashCode ajouter of avec pseudo comme valeur
  + Empêcher que le mot de passe soit tracer
    - Sur l’annotation @ToString ajouter of et lister les 3 champs
    - Autre solution vue un peu plus loin utiliser l’annotation @ToString.Exclude sur l’attribut password
      * Avantage, c’est que par défaut @ToString ne précise rien
      * Considère qu’il y a tous les attributs
      * Et @ToString.Exclude est posé sur les attributs à exclure

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

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

**import** lombok.\*;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

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

@EqualsAndHashCode(of = { "pseudo" })

@Entity

@Table(name = "CAV\_CLIENT")

@Builder

**public** **class** Client {

@Id

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

**private** String pseudo;

// @ToString.Exclude

@Column(name = "PASSWPORD", 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;

}

# Créer ClientRepository

**package** fr.eni.cave.dal;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** fr.eni.cave.bo.client.Client;

**public** **interface** ClientRepository **extends** JpaRepository<Client, String> {

}

# Créer des tests unitaires

* Créer une classe de tests unitaires annotées @DataJpaTest
* Créer 2 tests :
  + Sauver un client
  + Supprimer un client
  + Vos tests doivent tracer le client pour valider que le mot de passe ne s’affiche pas
  + Et utiliser au moins une fois le equals pour valider le comportement

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

**import** **static** org.assertj.core.api.Assertions.*assertThat*;

**import** **static** org.junit.jupiter.api.Assertions.*assertNull*;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.\*;

**import** fr.eni.cave.bo.client.Client;

**import** fr.eni.cave.dal.ClientRepository;

**import** lombok.extern.slf4j.Slf4j;

@Slf4j

@DataJpaTest

**public** **class** TestClient {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

ClientRepository repository;

@Test

**public** **void** test\_save() {

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.build();

// Appel du comportement

**final** Client clientDB = repository.save(client);

***log***.info(clientDB.toString());

// Vérification de la cascade de l'association

*assertThat*(clientDB).isNotNull();

*assertThat*(clientDB).isEqualTo(client);

}

@Test

**public** **void** test\_delete() {

**final** Client client = Client

.*builder*()

.pseudo("bobeponge@email.fr")

.password("carré")

.nom("Eponge")

.prenom("Bob")

.build();

// Contexte de la DB

entityManager.persist(client);

entityManager.flush();

***log***.info(client.toString());

// Appel du comportement

repository.delete(client);

// Vérification que l'entité a été supprimée

Client clientDB = entityManager.find(Client.**class**, client.getPseudo());

*assertNull*(clientDB);

}

}

* Les 2 tests unitaires valident la création de l’entité
  + La table créée pour l’entité Client :

create table cav\_client (passwpord varchar(68) not null, last\_name varchar(90) not null, first\_name varchar(150) not null, login varchar(255) not null, primary key (login))

* Voici la trace du client, il n’y a pas le mot de passe :

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

* Dans le test de sauvegarde, nous utilisons isEqualTo qui appelle la méthode equals