

Département d'Informatique

Filiere : IAAD

A ,U :2023-2024



جامعة مولاي إسماعيل
مَوْلَى إِسْمَاعِيلَ كُوٰنْجَيْ
UNIVERSITÉ MOULAY ISMAÏL



كلية العلوم
تَحْقِيقَاتِ وَالْعِلْمِ
FACULTÉ DES SCIENCES

Université Moulay Ismail faculté des Sciences Meknès

Département d'Informatique

TP N°2 : Spring Data JPA Hibernate

Module: Systèmes Distribués

Réalisé par :

- *Illa Meryeme*

Partie 1 :

On crée le projet maven students-app et on sélectionne les dépendances qu'on va utiliser (spring JPA, H2 database , Spring web et lombok)

On créer le modèle entities dans lequel on créer la classe Product pour gérer les produits

Cette classe est définie par ces attributs et quand 'on travaille avec Lombok on utilise La notation @data qui représentent les getters et les setters, Lombok permet de les ajoutés automatiquement

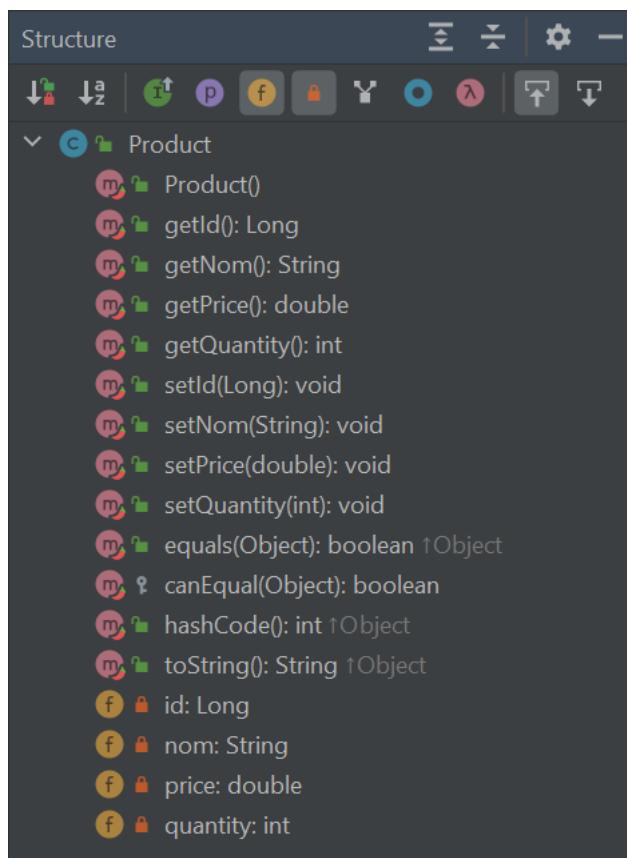
```
package com.example.studentsapp;

import lombok.Data;

@Data
public class Product {

    private Long id;
    private String nom;
    private double price;
    private int quantity;
}
```

La fenêtre structure



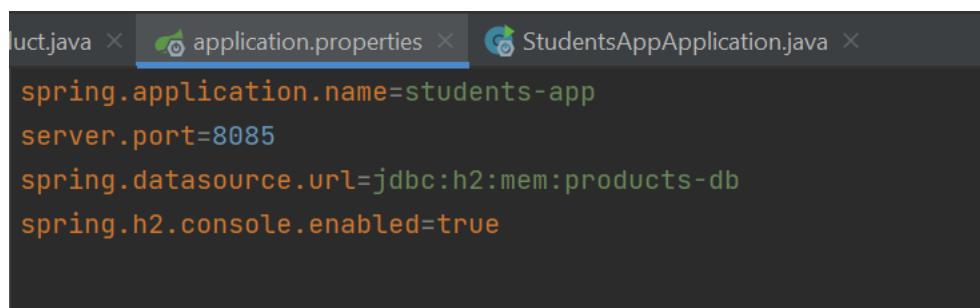
On crée une entité JPA avec la notation `@Entity` qui doit avoir un identifiant et on utilise la notation `@Id` et pour le générer automatiquement on utilise la notation `@GeneratedValue` avec la stratégie `IDENTITY`.

```
package com.example.studentsapp;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructorConstructor;
import lombok.Data;
import lombok.NoArgsConstructorConstructor;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Product {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private double price;
    private int quantity;
}
```

On spécifie les informations `server.port`, `spring.datasource.url` et la base de données `spring.h2.console.enabled` pour activer une interface web pour consulter la base de données dans le fichier `application.properties`



```
luct.java × application.properties × StudentsAppApplication.java ×
spring.application.name=students-app
server.port=8085
spring.datasource.url=jdbc:h2:mem:products-db
spring.h2.console.enabled=true
```

On démarre l'application SpringBoot et lorsque le démarrage, le premier qui va se démarrer c'est Spring et qu'on va le démarre il va faire l'inversion de contrôle par configurer JPA, scanner les classes et configurer les annotations

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure for 'students-app' with files like Product.java, application.properties, and StudentsAppApplication.java. The main editor window shows the code for StudentsAppApplication.java:

```
package com.example.studentsapp;
import ...
@SpringBootApplication
public class StudentsAppApplication {
    public static void main(String[] args) {
        SpringApplication.run(StudentsAppApplication.class, args);
    }
}
```

The bottom panel shows the run log for 'StudentsAppApplication':

```
... : NO active profile set, falling back to 1 default profile: "default"
... : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
... : Finished Spring Data repository scanning in 9 ms. Found 0 JPA repository interfaces.
... : Tomcat initialized with port 8085 (http)
... : Starting service [Tomcat]
... : Starting Servlet engine: [Apache Tomcat/10.1.19]
... : Initializing Spring embedded WebApplicationContext
... : HikariPool-1 - Starting...
```

On va se connecter à la base de données products

The screenshot shows the H2 Console login page. The URL is 'localhost:8085/h2-console/login.jsp?jsessionid=adfcc2e7c29ea31fafb80f10007ba2cb'. The page has a 'Login' header and fields for 'Saved Settings' (set to 'Generic H2 (Embedded)'), 'Setting Name' (set to 'Generic H2 (Embedded)'), 'Driver Class' ('org.h2.Driver'), 'JDBC URL' ('jdbc:h2:mem:products-db'), 'User Name' ('sa'), and 'Password'. There are 'Connect' and 'Test Connection' buttons at the bottom.

Important Commands

	Displays this Help Page
	Shows the Command History
	Executes the current SQL statement
	Executes the SQL statement defined by the text selection
	Auto complete
	Disconnects from the database

Sample SQL Script

```

Delete the table if it exists   DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns  CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row    INSERT INTO TEST VALUES(1, 'Hello');
Add another row  INSERT INTO TEST VALUES(2, 'World');
  
```

Pour exécuter le code une fois que spring est démarré la façon la plus simple c'est d'implémenter l'interface CommandLineRunner et on veut ajouter des produits à la base de données

Pour utiliser spring data on créer un package repository dans lequel on crée l'interface ProductRepository qui hérite de l'interface JpaRepository d'entité Product

```

package com.example.studentsapp.repository;

import com.example.studentsapp.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> {
}
  
```

On utilise l'interface ProductRepository pour pouvoir ajouter des produits dans la base de données et pour cela on déclare un objet de type ProductRepository et de faire l'injection des dépendances en utilisant la notation @Autowired et après on va enregistrer les produits puis les afficher dans la base de données

```

package com.example.studentsapp;
import com.example.studentsapp.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentsAppApplication implements CommandLineRunner {
    @Autowired
    private ProductRepository productRepository;
    public static void main(String[] args) {

        SpringApplication.run(StudentsAppApplication.class, args);
    }
    @Override
    public void run(String... args) throws Exception {
        productRepository.save(new Product( id: null, nom: "Computer", price: 4300, quantity: 3));
        productRepository.save(new Product( id: null, nom: "Printer", price: 1200, quantity: 4));
        productRepository.save(new Product( id: null, nom: "Smart Phone", price: 3200, quantity: 32));
    }
}

```

The screenshot shows the H2 Database Browser interface. On the left, there's a tree view of database objects:

- jdbc:h2:mem:products-db** (selected)
- PRODUCT** (table)
 - PRICE**
 - QUANTITY**
 - ID**
 - NOM**
 - Indexes**
- INFORMATION_SCHEMA**
- Users**
- H2 2.2.224 (2023-09-17)**

On the right, the SQL panel contains the query:

```
SELECT * FROM PRODUCT;
```

The results panel displays the following table:

PRICE	QUANTITY	ID	NOM
4300.0	3	1	Computer
1200.0	4	2	Printer
3200.0	32	3	Smart Phone

(3 rows, 5 ms)

Edit

On affiche une liste des produits

```

    @Override
    public void run(String... args) throws Exception {
        productRepository.save(new Product( id: null, nom: "Computer", price: 4300, quantity: 3));
        productRepository.save(new Product( id: null, nom: "Printer", price: 1200, quantity: 4));
        productRepository.save(new Product( id: null, nom: "Smart Phone", price: 3200, quantity: 32));
        List<Product> products = productRepository.findAll();
        products.forEach(p->
            System.out.println(p.toString());
        });
    }
}

```

The screenshot shows the IntelliJ IDEA interface with the code editor open. The code is annotated with line numbers from 17 to 29. The `run` method saves three products to a repository and then prints them to the console. The products are: Computer (id: null, nom: "Computer", price: 4300, quantity: 3), Printer (id: null, nom: "Printer", price: 1200, quantity: 4), and Smart Phone (id: null, nom: "Smart Phone", price: 3200, quantity: 32). The output window shows the printed product details.

On veut afficher le produit dont ID = 1

```

    Product product = productRepository.findById(Long.valueOf(1)).get();
    System.out.println("*****");
    System.out.println(product.getId());
    System.out.println(product.getNom());
    System.out.println(product.getQuantity());
    System.out.println("*****");
}
}

```

The screenshot shows the IntelliJ IDEA interface with the code editor open. The code is annotated with line numbers from 24 to 33. It finds a product by ID (1) and prints its details. The output window shows the printed product details.

H2 database est une base de données in memory c'est-à-dire c'est une base de données qui démarre en mémoire et une fois qu'on arrête l'application on perd les données et maintenant on veut basculer vers une base de données mySql

D'abord, on va changer dans les dépendances en ajoutant la dépendance mySql Driver dans le fichier pom.xml et la téléchargées et on a pris cette dépendance dans le site start.spring.io

```

    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>

```

Dans le fichier application.properties, on met la base de données de type mySql au lieu de la base de données de type H2

```

application.properties
spring.application.name=students-app
server.port=8085
spring.datasource.url=jdbc:mysql://localhost:3306/products-db?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect

```

On exécute l'application

	Base de données	Interclassement	Action
<input type="checkbox"/>	information_schema	utf8_general_ci	Vérifier les privilèges
<input type="checkbox"/>	mysql	utf8mb4_general_ci	Vérifier les privilèges
<input type="checkbox"/>	performance_schema	utf8_general_ci	Vérifier les privilèges
<input type="checkbox"/>	phpmyadmin	utf8_bin	Vérifier les privilèges
<input type="checkbox"/>	products-db	utf8mb4_general_ci	Vérifier les privilèges
<input type="checkbox"/>	test	latin1_swedish_ci	Vérifier les privilèges

Total : 6

	Éditer	Copier	Supprimer	id	nom	price	quantity
<input type="checkbox"/>				1	Computer	4300	3
<input type="checkbox"/>				2	Printer	1200	4
<input type="checkbox"/>				3	Smart Phone	3200	32

On réexécute une deuxième fois pour avoir le résultat car on a utilisé update c'est-à-dire à chaque fois il va insérer les produits lors de l'exécution

	Éditer	Copier	Supprimer	id	nom	price	quantity
<input type="checkbox"/>				1	Computer	4300	3
<input type="checkbox"/>				2	Printer	1200	4
<input type="checkbox"/>				3	Smart Phone	3200	32
<input type="checkbox"/>				4	Computer	4300	3
<input type="checkbox"/>				5	Printer	1200	4
<input type="checkbox"/>				6	Smart Phone	3200	32

Pour ajouter d'autres méthodes comme chercher un produit par exemple dont la désignation contient un mot clé, on a deux solutions :

- On ajoute dans l'interface une méthode qui retourne des produits et après sa déclaration on l'utilise dans l'application en cherchant tous les produits qui contiennent C

```

package com.example.studentsapp.repository;

import com.example.studentsapp.Product;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByNomContains(String mc);
}

```

```

System.out.println(product.getId());
System.out.println(product.getNom());
System.out.println(product.getQuantity());
System.out.println("*****");
System.out.println("-----");
List<Product> productList = productRepository.findByNomContains("C");
productList.forEach(product1 -> {
    System.out.println(product1);
});
}

```

Console

```

1
Computer
3
*****
-----
Product(id=1, nom=Computer, price=4300.0, quantity=3)
Product(id=4, nom=Computer, price=4300.0, quantity=3)
Product(id=7, nom=Computer, price=4300.0, quantity=3)

```

- On utilise une liste mais à condition d'ajouter la notation `@Query` et en utilisant la notation `@Param` pour indiquer que `mc` représente le paramètre `x` et on teste dans l'application

```

package com.example.studentsapp.repository;

import com.example.studentsapp.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByNomContains(String mc);

    @Query("select p from Product p where p.nom like :x")
    List<Product> search(@Param("x") String mc);
}

```

```
System.out.println(product1);
});
System.out.println("-----");
List<Product> productList1 = productRepository.search( mc: "%C%" );
productList1.forEach(p -> {
    System.out.println(p);
});
}

*****
Product(id=1, nom=Computer, price=4300.0, quantity=3)
Product(id=4, nom=Computer, price=4300.0, quantity=3)
Product(id=7, nom=Computer, price=4300.0, quantity=3)
Product(id=10, nom=Computer, price=4300.0, quantity=3)
-----
Product(id=1, nom=Computer, price=4300.0, quantity=3)
Product(id=4, nom=Computer, price=4300.0, quantity=3)
Product(id=7, nom=Computer, price=4300.0, quantity=3)
Product(id=10, nom=Computer, price=4300.0, quantity=3)
```

Selon les deux solutions, on trouve les mêmes résultats.

Maintenant on veut chercher les produits dont le prix est supérieur à 3000, on a 2 solutions :

- En utilisant la méthode `findByPriceGreaterThan()`

```
package com.example.studentsapp.repository;

import com.example.studentsapp.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByNomContains(String mc);
    List<Product> findByPriceGreaterThan(double price);

    @Query("select p from Product p where p.nom like :x")
    List<Product> search(@Param("x") String mc);
}
```

The screenshot shows the IntelliJ IDEA interface with the Product.java file open. The code prints a list of products from the database. Below the code, the terminal window shows the output:

```

Product(id=1, nom=Computer, price=4300.0, quantity=3)
Product(id=3, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=4, nom=Computer, price=4300.0, quantity=3)
Product(id=6, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=7, nom=Computer, price=4300.0, quantity=3)
Product(id=9, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=10, nom=Computer, price=4300.0, quantity=3)
Product(id=12, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=13, nom=Computer, price=4300.0, quantity=3)
Product(id=15, nom=Smart Phone, price=3200.0, quantity=32)

```

- En utilisant la fonction searchByPrice

The screenshot shows the ProductRepository.java code. It defines a repository interface extending JpaRepository. It includes methods for finding products by name and price, and two custom search methods using @Query annotations.

```

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {
    List<Product> findByNomContains(String mc);

    List<Product> findByPriceGreaterThan(double price);

    @Query("select p from Product p where p.nom like :x")
    List<Product> search(@Param("x") String mc);

    @Query("select p from Product p where p.price > :x")
    List<Product> searchByPrice(@Param("x") double price);
}

```

The screenshot shows the Product.java code with additional logic. It prints the first product, then prints a separator line, then finds products by price greater than 3000, prints them, then prints another separator line, then finds products by price 3000, prints them, and finally prints a third separator line. The terminal window shows the output:

```

Product(id=13, nom=Computer, price=4300.0, quantity=3)
Product(id=15, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=16, nom=Computer, price=4300.0, quantity=3)
Product(id=18, nom=Smart Phone, price=3200.0, quantity=32)

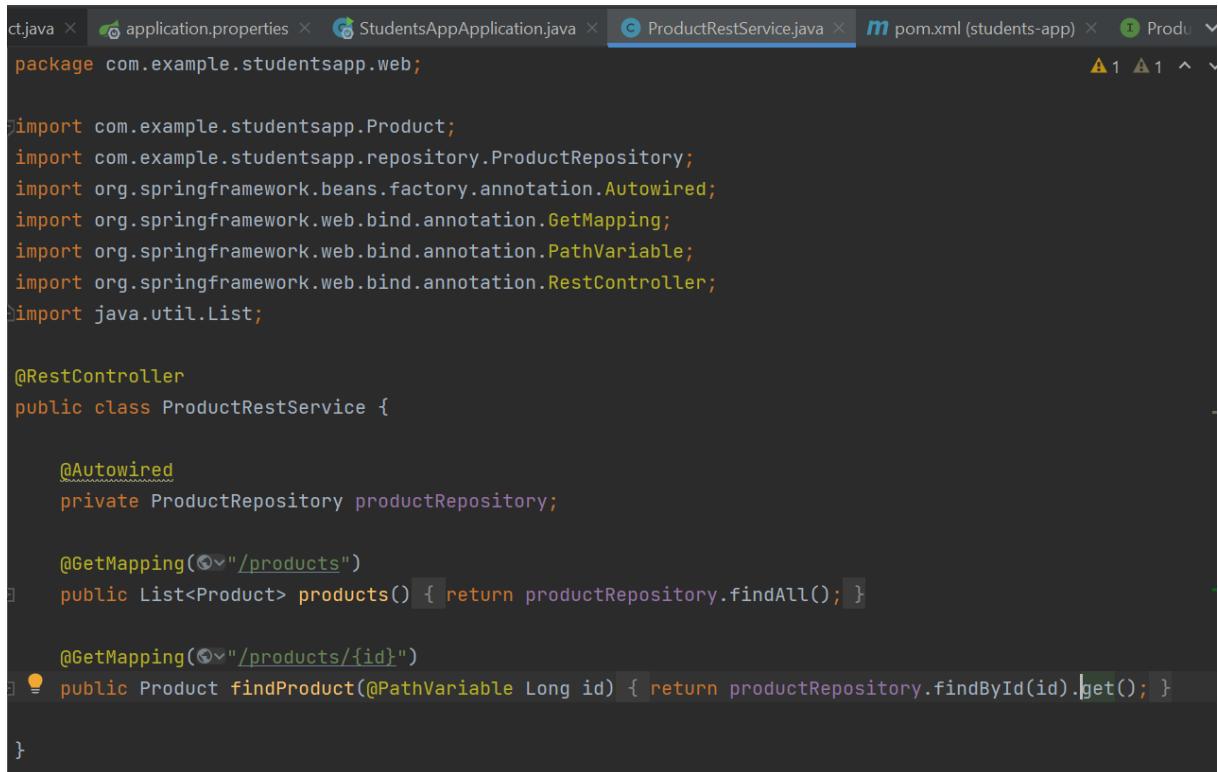
```

Les deux solutions donnent les mêmes résultats

```
Product(id=6, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=7, nom=Computer, price=4300.0, quantity=3)
Product(id=9, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=10, nom=Computer, price=4300.0, quantity=3)
Product(id=12, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=13, nom=Computer, price=4300.0, quantity=3)
Product(id=15, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=16, nom=Computer, price=4300.0, quantity=3)
Product(id=18, nom=Smart Phone, price=3200.0, quantity=32)
-----
Product(id=1, nom=Computer, price=4300.0, quantity=3)
Product(id=3, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=4, nom=Computer, price=4300.0, quantity=3)
Product(id=6, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=7, nom=Computer, price=4300.0, quantity=3)
Product(id=9, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=10, nom=Computer, price=4300.0, quantity=3)
Product(id=12, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=13, nom=Computer, price=4300.0, quantity=3)
Product(id=15, nom=Smart Phone, price=3200.0, quantity=32)
Product(id=16, nom=Computer, price=4300.0, quantity=3)
Product(id=18, nom=Smart Phone, price=3200.0, quantity=32)
```

Dans un package web, on crée la classe ProductRestService et pour créer un web service il suffit d'utiliser la notation @RestController et on a besoin d'accéder à la base de données c'est pour cela on déclare l'interface ProductRepository et on fait l'injection des dépendances par la notation @Autowired.

On crée une méthode qui permet de consulter la liste des produits et pour accéder à cette méthode, on utilise la notation @GetMapping (on envoie une requête http get vers le path « /products » et même chose pour une méthode qui permet de chercher un produit par son id mais au lieu d'utiliser findAll(), on utilise findById() et la notation @PathVariable pour indiquer que l'id va prendre la valeur qui vient dans la paramètre qui se trouve dans le path



```
package com.example.studentsapp.web;

import com.example.studentsapp.Product;
import com.example.studentsapp.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

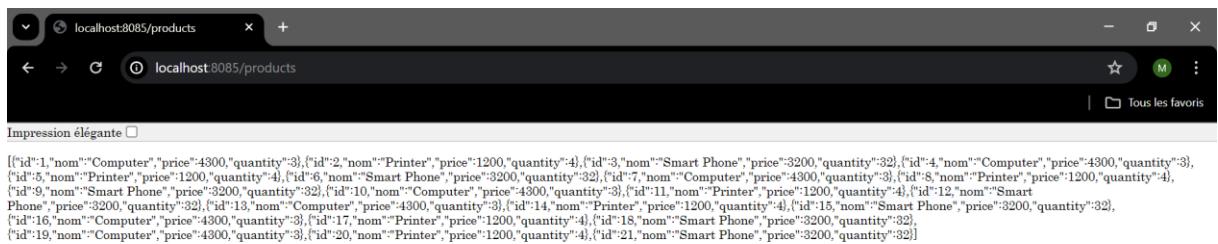
@RestController
public class ProductRestService {

    @Autowired
    private ProductRepository productRepository;

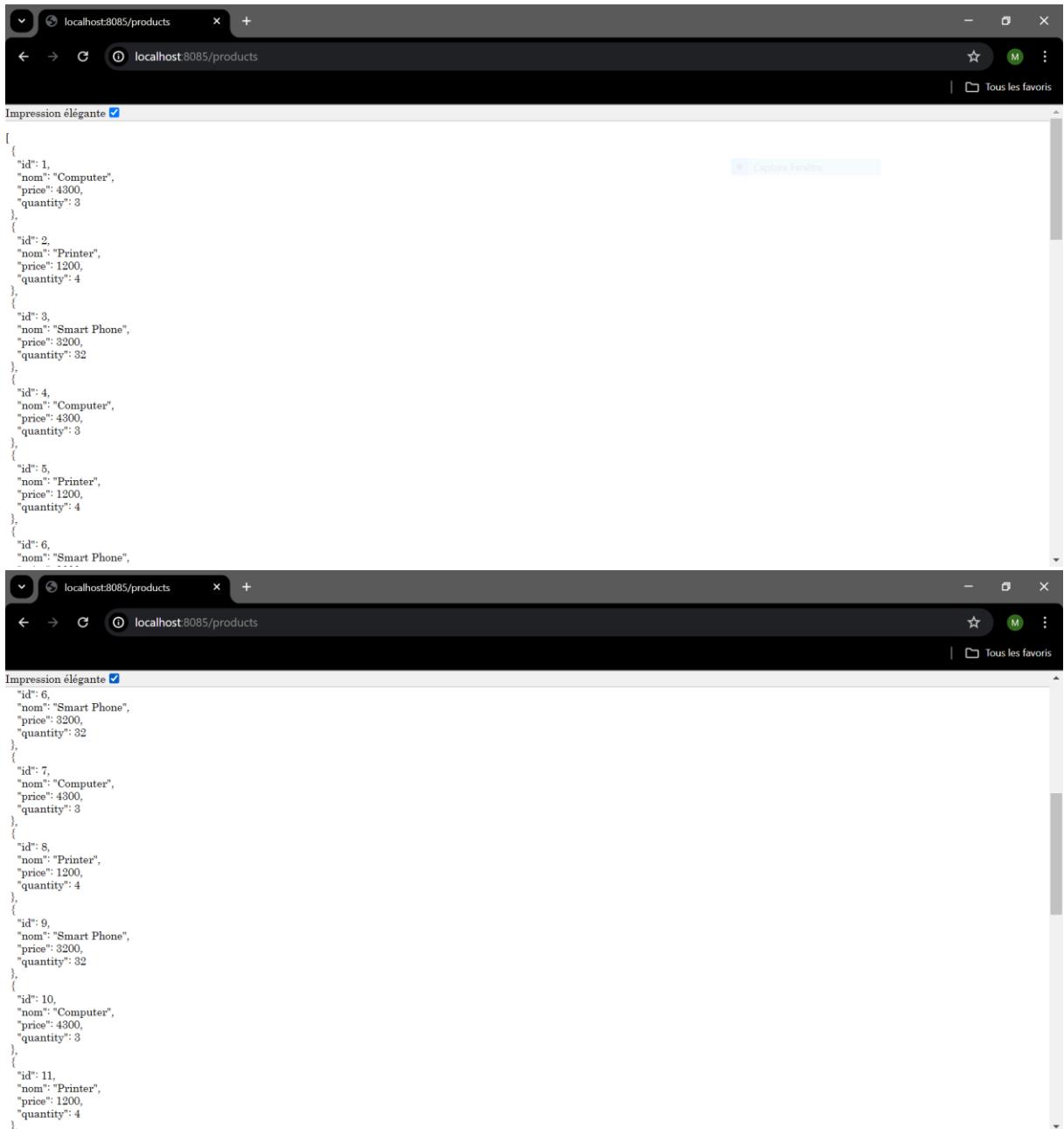
    @GetMapping("/products")
    public List<Product> products() { return productRepository.findAll(); }

    @GetMapping("/products/{id}")
    public Product findProduct(@PathVariable Long id) { return productRepository.findById(id).get(); }

}
```



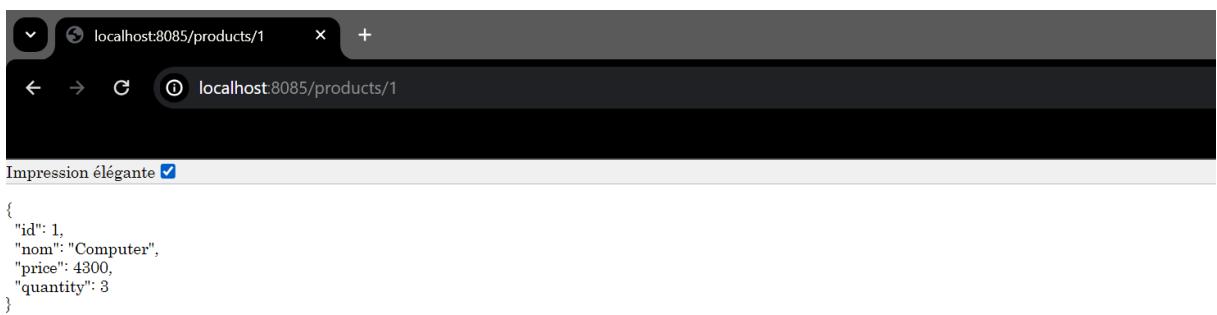
L'affichage avec une impression élégante



The image shows two separate browser windows side-by-side, both displaying the same JSON data for products. The URL in both tabs is `localhost:8085/products`.

```
[{"id": 1, "nom": "Computer", "price": 4300, "quantity": 3}, {"id": 2, "nom": "Printer", "price": 1200, "quantity": 4}, {"id": 3, "nom": "Smart Phone", "price": 3200, "quantity": 32}, {"id": 4, "nom": "Computer", "price": 4300, "quantity": 3}, {"id": 5, "nom": "Printer", "price": 1200, "quantity": 4}, {"id": 6, "nom": "Smart Phone", "price": 3200, "quantity": 32}, {"id": 7, "nom": "Computer", "price": 4300, "quantity": 3}, {"id": 8, "nom": "Printer", "price": 1200, "quantity": 4}, {"id": 9, "nom": "Smart Phone", "price": 3200, "quantity": 32}, {"id": 10, "nom": "Computer", "price": 4300, "quantity": 3}, {"id": 11, "nom": "Printer", "price": 1200, "quantity": 4}]
```

On veut afficher le produit dont l'ID est 1



The image shows a single browser window displaying the details of a specific product. The URL in the tab is `localhost:8085/products/1`.

```
{ "id": 1, "nom": "Computer", "price": 4300, "quantity": 3 }
```

Partie 2 :

On créer un projet maven hospital

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Dans un package entities, on crée les classes Patient, Medecin, Consultation, Rendez-vous et StatusRDV de type enum

Pour la date de naissance, on n'a pas besoin de garder l'heure, les minutes on utilise la notation @Temporal(TemporalType.DATE)

Pour la collection des rendezVous on utilise la notation @OneToMany

On créer une entité JPA avec La notation @Entity qui doit avoir un identifiant et on utilise la notation @Id et pour le générer automatiquement on utilise la notation @GeneratedValue avec la stratégie IDENTITY.

```
package ma.fsm.hospital.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Collection;
import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Consultation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateConsultation;
    private String rapport;
    @OneToOne
    private RendezVous rendezVous;

}
```

```
package ma.fsm.hospital.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Collection;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Medecin {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    private String specialite;
    @OneToMany(mappedBy = "medecin", fetch= FetchType.LAZY)
    private Collection<RendezVous> rendezVous;
}
```

```
package ma.fsm.hospital.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Collection;
import java.util.Date;

@Entity
@Data @NoArgsConstructor@AllArgsConstructor
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    private Date dateNaissance;
    private boolean malade;
    @OneToMany(mappedBy = "patient", fetch = FetchType.LAZY)
    private Collection<RendezVous> rendezVous;

}
```

```
package ma.fsm.hospital.entities;

public enum StatusRDV {
    PENDING,
    CANCELED,
    DONE
}
```

```
package ma.fsm.hospital.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class RendezVous {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date date;
    private StatusRDV status;
    private boolean annule;
    @ManyToOne
    private Patient patient;
    @ManyToOne
    private Medecin medecin;
    @OneToOne(mappedBy = "rendezVous")
    private Consultation consultation;
}
```

Dans un package repositories, on crée des interfaces pour chaque classe dans entities

Après on va faire des testes dans l'application avec l'implémentation de l'interface CommandLineRunner ou bien créer une méthode qui retourne un objet de type CommandLineRunner et utiliser la notation Bean

On va utiliser la deuxième méthode c'est mieux car elle nous a permis de faire l'injection sans utiliser la notation @Autowired

```

@SpringBootApplication
public class HospitalApplication {

    public static void main(String[] args) {
        SpringApplication.run(HospitalApplication.class, args);
    }

    @Bean
    CommandLineRunner start(PatientRepository patientRepository) {
        return args -> {
            Stream.of("Mohamed", "Hassan", "Najat")
                .forEach(name ->
                {
                    Patient patient = new Patient();
                    patient.setNom(name);
                    patient.setDateNaissance(new Date());
                    patient.setMalade(false);
                    patientRepository.save(patient);
                });
        };
    }
}

```

jdbc:h2:mem:hospital

- CONSULTATION
- MEDECIN
- PATIENT
- RENDEZ_VOUS
- INFORMATION_SCHEMA
- Users

H2 2.2.224 (2023-09-17)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PATIENT

DATE_NAISSANCE	MALADE	ID	NOM
2024-04-17	FALSE	1	Mohamed
2024-04-17	FALSE	2	Hassan
2024-04-17	FALSE	3	Najat

(3 rows, 44 ms)

Après on injecte encore l'interface MedecinRepository

```

CommandLineRunner start(PatientRepository patientRepository
, MedecinRepository medecinRepository) {
    return args -> {
        Stream.of("Mohamed", "Hassan", "Najat")
            .forEach(name ->
            {
                Patient patient = new Patient();
                patient.setNom(name);
                patient.setDateNaissance(new Date());
                patient.setMalade(false);
                patientRepository.save(patient);
            });
        Stream.of("Aymane", "Hanane", "Yasmine")
            .forEach(name ->
            {
                Medecin medecin = new Medecin();
                medecin.setNom(name);
                medecin.setEmail(name+"@gmail.com");
                medecin.setSpecialite(Math.random()>0.5?"Cardio":"Dentiste");
                medecinRepository.save(medecin);
            });
    };
}

```

The screenshot shows a Java code editor and a database query interface.

Java Code:

```

CommandLineRunner start(PatientRepository patientRepository
, MedecinRepository medecinRepository) {
    return args -> {
        Stream.of("Mohamed", "Hassan", "Najat")
            .forEach(name ->
            {
                Patient patient = new Patient();
                patient.setNom(name);
                patient.setDateNaissance(new Date());
                patient.setMalade(false);
                patientRepository.save(patient);
            });
        Stream.of("Aymane", "Hanane", "Yasmine")
            .forEach(name ->
            {
                Medecin medecin = new Medecin();
                medecin.setNom(name);
                medecin.setEmail(name+"@gmail.com");
                medecin.setSpecialite(Math.random()>0.5?"Cardio":"Dentiste");
                medecinRepository.save(medecin);
            });
    };
}

```

Database Query Tool:

- Toolbar: Auto commit checked, Max rows: 1000, Run Selected button.
- Left sidebar: Database connections (jdbc:h2:mem:hospital), tables: CONSULTATION, MEDECIN, PATIENT, RENDEZ_VOUS, INFORMATION_SCHEMA, Users.
- Bottom status: H2 2.2.224 (2023-09-17).
- SQL Statement input: `SELECT * FROM MEDECIN`.
- Result pane: A table showing the results of the query.

ID	EMAIL	NOM	SPECIALITE
1	Aymane@gmail.com	Aymane	Cardio
2	Hanane@gmail.com	Hanane	Cardio
3	Yasmine@gmail.com	Yasmine	Cardio

(3 rows, 0 ms)

Result:

```

SELECT * FROM MEDECIN;
+----+-----+-----+-----+
| ID | EMAIL | NOM  | SPECIALITE |
+----+-----+-----+-----+
| 1  | Aymane@gmail.com | Aymane | Cardio      |
| 2  | Hanane@gmail.com | Hanane | Cardio      |
| 3  | Yasmine@gmail.com | Yasmine | Cardio      |
+----+-----+-----+-----+
(3 rows, 0 ms)

```

On veut chercher un patient par son id premièrement et par son nom et un medecin par son nom et après on créer un rendezVous avec un patient et un medecin qui existe déjà

Dans la classe RendezVous, on utilise la notation @EnumType.STRING pour afficher les valeurs de type string et non pas de type ordinal

```

41             Medecin medecin = new Medecin();
42             medecin.setNom(name);
43             medecin.setEmail(name+"@gmail.com");
44             medecin.setSpecialite(Math.random()>0.5?"Cardio":"Dentiste");
45             medecinRepository.save(medecin);
46         });
47         Patient patient = patientRepository.findById(1L).orElse( other: null);
48         Patient patient1 = patientRepository.findByNom( name: "Mohamed");
49
50         Medecin medecin= medecinRepository.findByNom("Yasmine");
51         RendezVous rendezVous= new RendezVous();
52         rendezVous.setDate(new Date());
53         rendezVous.setStatus(StatusRDV.PENDING);
54         rendezVous.setMedecin(medecin);
55         rendezVous.setPatient(patient);
56         rendezVousRepository.save(rendezVous);
57
58     };
59 }
60
61

```

Init Max rows: 1000 | Auto complete Off Auto select On

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM RENDEZ_VOUS
```

HÉMA

)

SELECT * FROM RENDEZ_VOUS;					
ANNULE	DATE	ID	MEDECIN_ID	PATIENT_ID	STATUS
FALSE	2024-04-17 12:58:51.269	1	3	1	PENDING

(1 row, 2 ms)

```

        medecin.setSpecialite("Orthopédie")>>0.5: Consultation , 
        medecinRepository.save(medecin);
    });

Patient patient = patientRepository.findById(1L).orElse( other: null);
Patient patient1 = patientRepository.findByNom( name: "Mohamed");

Medecin medecin= medecinRepository.findByNom("Yasmine");
RendezVous rendezVous= new RendezVous();
rendezVous.setDate(new Date());
rendezVous.setStatus(StatusRDV.PENDING);
rendezVous.setMedecin(medecin);
rendezVous.setPatient(patient);
rendezVousRepository.save(rendezVous);
RendezVous rendezVous1 =rendezVousRepository.findById(1L).orElse( other: null);
Consultation consultation= new Consultation();
consultation.setDateConsultation(new Date());
consultation.setRendezVous(rendezVous1);
consultation.setRapport("Rapport de la consultation...");
consultationRepository.save(consultation);

};

}
}

```

SELECT * FROM CONSULTATION |

SELECT * FROM CONSULTATION;			
DATE_CONSULTATION	ID	RENDEZ_VOUS_ID	RAPPORT
2024-04-17 13:08:11.058	1	1	Rapport de la consultation...

(1 row, 2 ms)

[Edit](#)

On crée une interface dans un package service dans laquelle on ajoute quelque méthode dont on aurait besoin

On utilise la notation @Service (utilisée pour les objets de la couche métier) et la notation @Transactional pour que toutes les méthodes soit transactionnelles

Pour injecter les dépendances on utilise le constructeur de toutes les paramètres

L'identifiant Id n'est pas nécessairement toujours de type Long, il peut être une chaîne de caractère.
En utilisant une méthode pour générer une chaîne de caractère aléatoire unique

```
package ma.fsm.hospital.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class RendezVous {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date date;
    private StatusRDV status;
    private boolean annule;
    @ManyToOne
    private Patient patient;
    @ManyToOne
    private Medecin medecin;
    @OneToOne(mappedBy = "rendezVous")
    private Consultation consultation;
}
```

```
package ma.fsm.hospital.repositories;

import ma.fsm.hospital.entities.RendezVous;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RendezVousRepository extends JpaRepository<RendezVous, String> {
}
```

```

Medecin medecin= medecinRepository.findByNom("Yasmine");
RendezVous rendezVous= new RendezVous();
rendezVous.setDate(new Date());
rendezVous.setStatus(StatusRDV.PENDING);
rendezVous.setMedecin(medecin);
rendezVous.setPatient(patient);
hospitalService.saveRDV(rendezVous);
RendezVous saveDRDV = hospitalService.saveRDV(rendezVous);
System.out.println(saveDRDV.getId());
RendezVous rendezVous1 = rendezVousRepository.findAll().get(0);
Consultation consultation= new Consultation();
consultation.setDateConsultation(new Date());
consultation.setRendezVous(rendezVous1);
consultation.setRapport("Rapport de la consultation...");
hospitalService.saveConsultation(consultation);

};

}
}

```

SQL statement:

SELECT * FROM RENDEZ_VOUS|

SELECT * FROM RENDEZ_VOUS;

ANNULE	DATE	MEDECIN_ID	PATIENT_ID	ID	STATUS
FALSE	2024-04-17 16:20:33.459	3	1	7018bfe1-00b8-44d1-89a4-58efbba8225f	PENDING
FALSE	2024-04-17 16:20:33.459	3	1	1fc6599a-12f3-4797-a80a-c45b3d533bf9	PENDING

(2 rows, 4 ms)

La partie web :

Dans un package web, on crée la classe PatientRestService et pour créer un web service il suffit d'utiliser la notation @RestController et on fait l'injection des dépendances par la notation @Autowired.

On veut consulter la liste des patients et on trouve un problème de dépendances cycliques car on a une relation bidirectionnelle donc pour le résoudre on utilise la notation JsonProperty pour prendre en considération l'attribut lorsqu'on fait l'ajout mais non pas dans la lecture

```
import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class RendezVous {
    @Id
    private String id;
    private Date date;
    @Enumerated(EnumType.STRING)
    private StatusRDV status;
    private boolean annule;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Patient patient;
    @ManyToOne
    private Medecin medecin;
    @OneToOne(mappedBy = "rendezVous")
    private Consultation consultation;
}
```

```
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Collection;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Medecin {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    private String specialite;
    @OneToMany(mappedBy = "medecin", fetch= FetchType.LAZY)
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Collection<RendezVous> rendezVous;
}
```

```

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Collection;
import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Consultation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateConsultation;
    private String rapport;
    @OneToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private RendezVous rendezVous;
}

}

```

```

package ma.fsm.hospital.web;

import ma.fsm.hospital.entities.Patient;
import ma.fsm.hospital.repositories.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
public class PatientRestController {
    @Autowired
    private PatientRepository patientRepository;
    @GetMapping("/patients")
    public List<Patient> patientList() { return patientRepository.findAll(); }
}

```

Impression élégante □

```
[{"id":1,"nom":"Mohamed","dateNaissance":"2024-04-17","malade":false,"rendezVous":[{"id":119f620a7ad14ca5b6520114c306cd85,"date":"2024-04-17T14:55:52.694+00:00","status":"PENDING","annule":false,"medecin":{"id":3,"nom":"Yasmine","email":"Yasmine@gmail.com","specialite":"Cardio"},"consultation":{"id":1,"dateConsultation":"2024-04-17T14:55:52.719+00:00","rapport":"Rapport de la consultation..."}, {"id":5a76238e408c41a0ac87b50507f2a951,"date":"2024-04-17T14:55:52.694+00:00","status":"PENDING","annule":false,"medecin":{"id":3,"nom":"Yasmine","email":"Yasmine@gmail.com","specialite":"Cardio"},"consultation":null}], "id":2,"nom":"Hassan","dateNaissance":"2024-04-17","malade":false,"rendezVous":[]}, {"id":3,"nom":"Najat","dateNaissance":"2024-04-17","malade":false,"rendezVous":[]}]
```

Partie 3 :

Dans cette partie, on fait la même chose pour les deux premières parties

On utilise la notation @Column(unique= true, length=20) sur un colonne pour dire qu'il est unique et il ne peut dépasser 20 et il peut donner un nom

On utilise la notation @Service pour la couche service et la notation @Transactional pour gérer les transactions

On utilise la notation @Repository pour indiquer que ce component de la couche dao

On fait l'injection des constructeurs

Package entities :

```
package ma.fsm.jpafsm.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Role {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String desc;
    @Column(length = 20, unique = true)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
    // @JoinTable(name = "USERS_ROLES", )
    private List<User> users= new ArrayList<>();

}
```

```
package ma.fsm.jpafsm.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;
@Entity
@Table(name= "USERS")
@Data @NoArgsConstructor@AllArgsConstructor
public class User {
    @Id
    private String userId;
    @Column(name = "USER_NAME",unique = true, length = 20)
    private String username;
    private String password;
    @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
    private List<Role> roles = new ArrayList<>();

}
}
```

Package repositories :

```
package ma.fsm.jpafsm.repositories;

import ma.fsm.jpafsm.entities.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findByRoleName(String rolename);
}
```

```
package ma.fsm.jpafsm.repositories;

import ma.fsm.jpafsm.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, String> {
    User findByUsername(String username);
}
```

Package service :

```
package ma.fsm.jpafsm.service;

import ma.fsm.jpafsm.entities.Role;
import ma.fsm.jpafsm.entities.User;

public interface UserService {
    User addNewUser(User user);
    Role addNewRole(Role role);
    User findUserByUserName(String userName);
    Role findRoleByRoleName(String roleName);
    void addRoleToUser(String username, String rolename);
}
```

```
@Service
@Transactional
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private UserRepository userRepository;
    private RoleRepository roleRepository;
    @Override
    public User addNewUser(User user) {
        user.setUserId(UUID.randomUUID().toString());
        return userRepository.save(user);
    }
    @Override
    public Role addNewRole(Role role) { return null; }
    @Override
    public User findUserByUserName(String userName) {
        return userRepository.findByUsername(userName);
    }
    @Override
    public Role findRoleByRoleName(String roleName) {
        return roleRepository.findByName(roleName);
    }
    @Override
    public void addRoleToUser(String username, String rolename) {
        User user = findUserByUserName(username);
        Role role = findRoleByRoleName(rolename);
        if (user.getRoles() != null)
            user.getRoles().add(role);
    }
}
```

On spécifie la base de données

```
spring.application.name=jpa-fsm
spring.datasource.url=jdbc:h2:mem:users_db
server.port=8083
spring.h2.console.enabled=true
```

L'application :

```

        return args -> {
            User u= new User();
            u.setUsername("user1");
            u.setPassword("123456");
            userService.addNewUser(u);

            User u2= new User();
            u2.setUsername("admin");
            u2.setPassword("123456");
            userService.addNewUser(u2);

            Stream.of("STUDENT", "USER", "ADMIN").forEach(r ->{
                Role role1= new Role();
                role1.setRoleName(r);
                userService.addNewRole(role1);
            });
            userService.addRoleToUser( username: "user1", rolename: "STUDENT");
            userService.addRoleToUser( username: "user1", rolename: "USER");
            userService.addRoleToUser( username: "admin", rolename: "USER");
            userService.addRoleToUser( username: "admin", rolename: "ADMIN");
            Role role1= new Role();
            role1.setRoleName("STUDENT");
            userService.addNewRole(role1);
        };
    };
}

```

Screenshot of a database management tool showing the results of a query on the 'ROLE' table.

The interface includes:

- Toolbar with icons for connection, refresh, auto-commit (checked), max rows (set to 1000), run, auto-complete (off), and auto-select (on).
- Left sidebar showing database schema: jdbc:h2:mem:users_db, ROLE, ROLE_USERS, USERS, INFORMATION_SCHEMA, and Users.
- Bottom status bar: H2 2.2.224 (2023-09-17).
- SQL statement input field: `SELECT * FROM ROLE`.
- Result table output:

ID	ROLE_NAME	DESC
1	STUDENT	null
2	USER	null
3	ADMIN	null

(3 rows, 3 ms)

[Edit](#)

The screenshot shows two separate database sessions in the H2 Database Browser.

Top Window (Database: users_db):

- SQL Statement: `SELECT * FROM USERS;`
- Table Data:

USER_NAME	PASSWORD	USER_ID
user1	123456	29498160-88ed-4058-84fc-5918dbc8b74d
admin	123456	0d1de21e-8c44-449b-9a96-7ea93653a47f

- (2 rows, 1 ms)
- Buttons: Run, Run Selected, Auto complete, Clear, SQL statement: `SELECT * FROM USERS;`

Bottom Window (Database: users_db):

- SQL Statement: `SELECT * FROM ROLE_USERS;`
- Table Data:

ROLES_ID	USERS_USER_ID
1	29498160-88ed-4058-84fc-5918dbc8b74d
2	29498160-88ed-4058-84fc-5918dbc8b74d
2	0d1de21e-8c44-449b-9a96-7ea93653a47f
3	0d1de21e-8c44-449b-9a96-7ea93653a47f

- (4 rows, 0 ms)
- Buttons: Run, Run Selected, Auto complete, Clear, SQL statement: `SELECT * FROM ROLE_USERS;`

On crée un modèle d'authentification

On utilise la notation `@ToString.Exclude` sur l'attribut de la liste des utilisateurs dans la classe Rôle pour spécifier à lombok que ce n'est pas la peine d'inclure la méthode `toString`

```
}

@Override
public User authenticate(String userName, String password) {
    User user= userRepository.findByUsername(userName);
    if(user == null)    throw new RuntimeException("Bad credentials");
    if(user.getPassword().equals(password)) {
        return user;
    }
    throw new RuntimeException("Bad credentials");
}
}
```

```
Role role1 = new Role();
role1.setRoleName("STUDENT");
// userService.addNewRole(role1);
try {
    User user = userService.authenticate( userName: "user1",   password: "123456");
    System.out.println(user.getUserId());
    System.out.println(user.getUsername());
    user.getRoles().forEach(r -> {
        System.out.println("Roles =>" + r.toString());
    });
} catch (Exception e) {
    e.printStackTrace();
}
};

}

}
```

```

import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.ArrayList;
import java.util.List;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Role {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String desc;
    @Column(length = 20, unique = true)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
    // @JoinTable(name = "USERS_ROLES", )
    @ToString.Exclude
    private List<User> users= new ArrayList<>();
}

```

```

2024-04-18T22:31:18.454+02:00 INFO 7536 --- [jpa-fsm] [           main] org.hibernate.Hibernate: HHH000284: Processing PersistenceUnitInfo
2024-04-18T22:31:18.469+02:00 INFO 7536 --- [jpa-fsm] [           main] o.h.c.integrator.PersistenceUnitIntegrator: HHH000412: Hibernate ORM
2024-04-18T22:31:18.607+02:00 INFO 7536 --- [jpa-fsm] [           main] o.s.o.j.p.PersistenceUnitProperties: HHH000026: Second-level
2024-04-18T22:31:19.202+02:00 INFO 7536 --- [jpa-fsm] [           main] o.h.e.t.j.jta.JtaPlatformInitiator: HHH000489: No JTA platform
2024-04-18T22:31:19.242+02:00 INFO 7536 --- [jpa-fsm] [           main] j.LocalContainerEntityManagerFactoryBean: JpaBaseConfiguration
2024-04-18T22:31:19.525+02:00 WARN 7536 --- [jpa-fsm] [           main] JpaBaseConfiguration: spring.jpa.open-in-view
2024-04-18T22:31:19.755+02:00 INFO 7536 --- [jpa-fsm] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer: Tomcat started on port 8080
2024-04-18T22:31:19.759+02:00 INFO 7536 --- [jpa-fsm] [           main] ma.fsm.jpafsm.JpaFsmApplication: Started JpaFsmApplication
29f7e9d0-5ccf-42c0-b9b5-d93426a12b93
user1
Roles =>Role(id=1, desc=null, roleName=STUDENT)
Roles =>Role(id=2, desc=null, roleName=USER)
|
```

On essaie avec un mot de passe incorrect, on trouve :

```

2024-04-18T22:42:45.032+02:00 INFO 16928 --- [jpa-fsm] [           main] o.hibernate.jpa.internal.util.LogHelper : HHH000284: Processing Pe
2024-04-18T22:42:45.070+02:00 INFO 16928 --- [jpa-fsm] [           main] org.hibernate.Version : HHH000412: Hibernate OR
2024-04-18T22:42:45.092+02:00 INFO 16928 --- [jpa-fsm] [           main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level
2024-04-18T22:42:45.268+02:00 INFO 16928 --- [jpa-fsm] [           main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup
2024-04-18T22:42:45.841+02:00 INFO 16928 --- [jpa-fsm] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platfo
2024-04-18T22:42:45.882+02:00 INFO 16928 --- [jpa-fsm] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManager
2024-04-18T22:42:46.173+02:00 WARN 16928 --- [jpa-fsm] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view
2024-04-18T22:42:46.391+02:00 INFO 16928 --- [jpa-fsm] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8
2024-04-18T22:42:46.398+02:00 INFO 16928 --- [jpa-fsm] [           main] ma.fsm.jpafsm.JpaFsmApplication : Started JpaFsmApplication
java.lang.RuntimeException Create breakpoint : Bad credentials
at ma.fsm.jpafsm.service.UserServiceImpl.authenticate(UserServiceImpl.java:53) <2 internal lines>
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:351)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:196)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
```

On va basculer vers mySql

Sur mysql, on trouve des exceptions quand il essaie de créer la base de données il trouve le champ desc

```
6524 --- [jpa-fsm] [main] o.h.e.t.j.p.i.JtaPlatformInitiator      : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to 'true' or use a JTA provider)
```

```
6524 --- [jpa-fsm] [main] o.h.t.s.i.ExceptionHandlerLoggedImpl : GenerationTarget encountered exception accepting command : Error
```

```
ndAcceptanceException Create breakpoint : Error executing DDL "create table role (id bigint not null auto_increment, desc varchar(255), role_name varc
forEach(HashMap.java:1429) ~[na:na] <6 internal lines>
vendor.SpringHibernateJpaPersistenceProvider.createContainerEntityManagerFactory(SpringHibernateJpaPersistenceProvider.java:73) ~[spring-orm-6.1.5.jar:6.1.5]
LocalContainerEntityManagerFactoryBean.createNativeEntityManagerFactory(LocalContainerEntityManagerFactoryBean.java:390) ~[spring-orm-6.1.5.jar:6.1.5]
AbstractEntityManagerFactoryBean.buildNativeEntityManagerFactory(AbstractEntityManagerFactoryBean.java:409) ~[spring-orm-6.1.5.jar:6.1.5]
AbstractEntityManagerFactoryBean.afterPropertiesSet(AbstractEntityManagerFactoryBean.java:396) ~[spring-orm-6.1.5.jar:6.1.5]
LocalContainerEntityManagerFactoryBean.afterPropertiesSet(LocalContainerEntityManagerFactoryBean.java:366) ~[spring-orm-6.1.5.jar:6.1.5]
ctory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1833) ~[spring-beans-6.1.5.jar:6.1.5]
ctory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1782) ~[spring-beans-6.1.5.jar:6.1.5]
ctory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:600) ~[spring-beans-6.1.5.jar:6.1.5]
ctory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:522) ~[spring-beans-6.1.5.jar:6.1.5]
```

Pour résoudre ce problème on utilise la notation @Column(name= « DESCRIPTION »)

```
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.ArrayList;
import java.util.List;
@Entity
@NoArgsConstructor @AllArgsConstructor
public class Role {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "DESCRIPTION")
    private String desc;
    @Column(length = 20, unique = true)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
    // @JoinTable(name = "USERS_ROLES", )
    @ToString.Exclude
    private List<User> users= new ArrayList<>();

}
```

L'affichage :

	Éditer	Copier	Supprimer	user_id	password	user_name
<input type="checkbox"/>				82f8dc4d-4926-4161-81f8-3c1b49b17df5	123456	admin
<input type="checkbox"/>				c7761e99-8449-436b-ae1a-8516180cfe05	123456	user1

	<input type="button" value="← T →"/>			id	description	role_name
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	NULL	STUDENT
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	NULL	USER
<input type="checkbox"/>	Éditer	Copier	Supprimer	3	NULL	ADMIN

roles_id	users_user_id
1	c7761e99-8449-436b-ae1a-8516180cf05
2	c7761e99-8449-436b-ae1a-8516180cf05
2	82f8dc4d-4926-4161-81f8-3c1b49b17df5
3	82f8dc4d-4926-4161-81f8-3c1b49b17df5

On réalise un contrôle pour consulter les utilisateurs, on va créer un contrôleur

Package web :

```
import ma.fsm.jpafsm.entities.User;
import ma.fsm.jpafsm.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {
    @Autowired
    private UserService userService;
    @GetMapping("/users/{username}")
    public User user(@PathVariable String username) {
        User user= userService.findUserByUserName(username);
        return user;
    }
}
```

```
import java.util.List;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;
@Entity
@NoArgsConstructor @AllArgsConstructor
public class Role {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "DESCRIPTION")
    private String desc;
    @Column(length = 20, unique = true)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
    // @JoinTable(name = "USERS_ROLES", )
    @ToString.Exclude
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<User> users= new ArrayList<>();
}
```

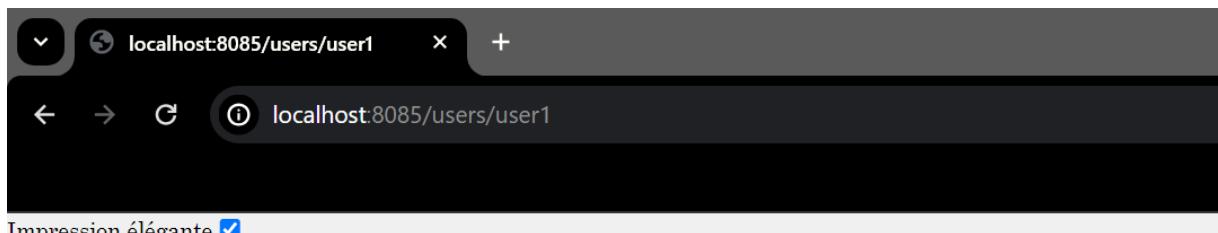
```

import java.util.ArrayList;
import java.util.List;
@Entity
@Table(name= "USERS")
@Data @NoArgsConstructor @AllArgsConstructor
public class User {
    @Id
    private String userId;
    @Column(name = "USER_NAME",unique = true, length = 20)
    private String username;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;
    @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
    private List<Role> roles = new ArrayList<>();

}

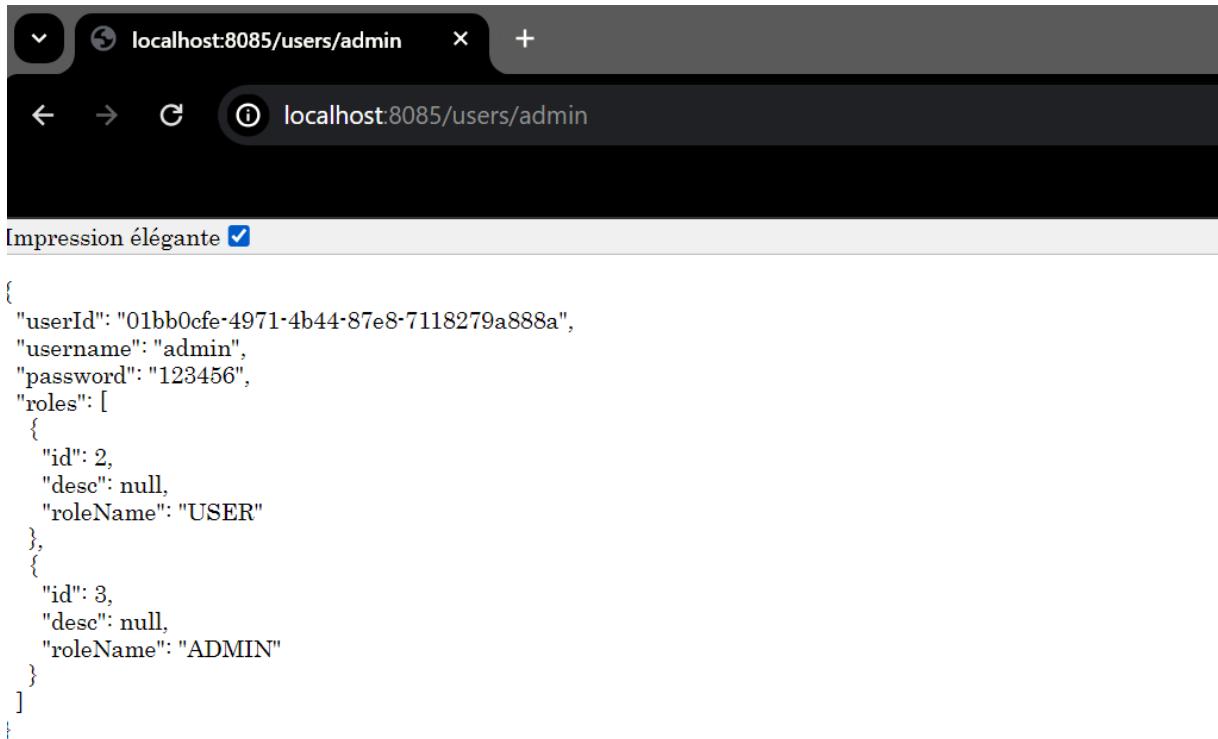
```

L'affichage :



Impression élégante

```
{
  "userId": "689a8ab8-9714-4145-ac5e-2f600e0de07b",
  "username": "user1",
  "password": "123456",
  "roles": [
    {
      "id": 1,
      "desc": null,
      "roleName": "STUDENT"
    },
    {
      "id": 2,
      "desc": null,
      "roleName": "USER"
    }
  ]
}
```



A screenshot of a web browser window titled "localhost:8085/users/admin". The address bar also shows "localhost:8085/users/admin". Below the address bar, there is a message "Impression élégante" with a checked checkbox. The main content area displays a JSON object representing a user:

```
{  
  "userId": "01bb0cfe-4971-4b44-87e8-7118279a888a",  
  "username": "admin",  
  "password": "123456",  
  "roles": [  
    {  
      "id": 2,  
      "desc": null,  
      "roleName": "USER"  
    },  
    {  
      "id": 3,  
      "desc": null,  
      "roleName": "ADMIN"  
    }  
  ]  
}
```