Création d’une API CRUD en Java

avec Spring Boot, Maven, H2, et des tests via curl

PARTIE 1 : CRUD simple

🧱 1. Structure de projet Maven

Initialiser les dépendances sous https://start.spring.io/

Spring Web	Créer une API REST (@RestController, routes HTTP, etc.)
Spring Data JPA	Mapper les entités avec la base de données via JPA/Hibernate
H2 Database	Fournir une base de données en mémoire pour le développement

Facultatif mais recommandées :

Spring Boot DevTools	Recharge automatique à chaud pour faciliter le dev
Springdoc OpenAPI WebMVC UI	Génère l'interface Swagger

```
product-api/
 ├── src/
 │    ├── main/
 │    │    ├── java/com/example/productapi/
 │    │    │    ├── ProductApiApplication.java
 │    │    │    ├── controller/ProductController.java
 │    │    │    ├── model/Product.java
 │    │    │    └── repository/ProductRepository.java
 │    └── resources/
 │         ├── application.properties
 ├── pom.xml
```

📦 2. pom.xml (Maven)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>product-api</artifactId>
```

```
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <java.version>17</java.version>
    <spring.boot.version>3.2.0</spring.boot.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

📄 3. Entité Product.java

```
package com.example.productapi.model;
import jakarta.persistence.*;

@Entity
public class Product {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private double price;

    // Getters et setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
}
```

🗃️ 4. Repository ProductRepository.java

```
package com.example.productapi.repository;

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

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

🌐 5. Contrôleur ProductController.java

```
package com.example.productapi.controller;

import com.example.productapi.model.Product;
import com.example.productapi.repository.ProductRepository;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/products")
public class ProductController {

    private final ProductRepository repository;

    public ProductController(ProductRepository repository) {
```

```
        this.repository = repository;
    }

    @GetMapping
    public List<Product> getAll() {
        return repository.findAll();
    }

    @GetMapping("/{id}")
    public Product getById(@PathVariable Long id) {
        return repository.findById(id).orElseThrow();
    }

    @PostMapping
    public Product create(@RequestBody Product product) {
        return repository.save(product);
    }

    @PutMapping("/{id}")
    public Product update(@PathVariable Long id, @RequestBody Product product) {
        Product existing = repository.findById(id).orElseThrow();
        existing.setName(product.getName());
        existing.setPrice(product.getPrice());
        return repository.save(existing);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        repository.deleteById(id);
    }
}
```

⚙️ 6. application.properties

le **fichier de configuration principal** d'une application Spring Boot. Il permet de **surcharger les valeurs par défaut** fournies par Spring Boot, comme :

- les propriétés de connexion à la base de données,
- le port du serveur,
- le comportement de JPA/Hibernate,
- l’activation de la console H2,
- le logging, etc.

```
# Connexion à H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Console H2 activée http://localhost:8080/h2-console
spring.jpa.show-sql=true

# JPA & Hibernate
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
```

🚀 7. Lancement de l'application

ProductApiApplication.java :

```
package com.example.productapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ProductApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProductApiApplication.class, args);
    }
}
```

Compiler et exécuter avec :

```
./mvnw spring-boot:run
```

🧪 8. Tests avec curl

Créer :

```
curl -X POST -H "Content-Type: application/json" \
-d '{"name":"Stylo","price":2.5}' \
http://localhost:8080/products
```

Lister :

curl http://localhost:8080/products

Afficher un produit :

curl http://localhost:8080/products/1

Mettre à jour :

curl -X PUT -H "Content-Type: application/json" \
-d '{"name":"Stylo bleu","price":2.7}' \
http://localhost:8080/products/1

Supprimer :

curl -X DELETE http://localhost:8080/products/1

PARTIE 2 : Méthodes personnalisées

🧪 1. Ajouter une méthode DUPLICATE

La méthode DUPLICATE doit permettre de **cloner un produit existant** à partir de son id.
Créer une nouvelle route HTTP (par exemple : /products/{id}/duplicate) qui :

- récupère le produit d'origine via son id,
- copie ses propriétés (name, price),
- sauvegarde un **nouveau produit** en base,
- renvoie ce nouveau produit.

Résultat attendu :
Si on a en base

SELECT * FROM PRODUCT;

ID	NAME	PRICE
1	Clavier	40.0

(1 row, 1 ms)

Après duplication :

SELECT * FROM PRODUCT;

ID	NAME	PRICE
1	Clavier	40.0
2	Clavier (copie)	40.0

(2 rows, 0 ms)

Vérifier le fonctionnement avec une méthode curl.

🌐 2. Ajouter une fonction BUNDLE

On souhaite créer une **fonction "bundle" intelligente**, avec :

1. Un nouveau produit construit à partir d'un ensemble d'autres produits,
   Par exemple, j'ai trois produits en base : 1-'ordinateur', 2-'clavier', 3-'moniteur'. Je peux créer un bundle 4-'ordinateur + clavier + moniteur'.
2. Une table de liaison sourceIds pour savoir quels produits composent ce bundle,
3. Une **protection contre les boucles** (ex : un produit A qui contient B, qui contient A → interdit).

🧱 Étapes globales

1. 💾 Modifier l'entité **Product** pour avoir la relation vers ses sources :

```
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private double price;

    @ManyToMany
    @JoinTable(
        name = "product_sources",
        joinColumns = @JoinColumn(name = "product_id"),
        inverseJoinColumns = @JoinColumn(name = "source_id")
    )
    private List<Product> sources = new ArrayList<>();

    // Getters et setters
```

2. ➕ Créer la méthode **createBundle** dans le contrôleur

3. 🔁 Vérifier la non-récursivité

4. ✅ Ajouter un test CURL pour vérifier

Supposons qu'on ait déjà en base :

```
curl -X POST http://localhost:8080/products \
-H "Content-Type: application/json" \
-d '{"name":"Ordinateur","price":500}'

curl -X POST http://localhost:8080/products \
-H "Content-Type: application/json" \
-d '{"name":"Clavier","price":50}'

curl -X POST http://localhost:8080/products \
-H "Content-Type: application/json" \
-d '{"name":"Moniteur","price":200}'
```

Création du bundle :

```
curl -X POST http://localhost:8080/products/bundle \
-H "Content-Type: application/json" \
-d '[1,3,4]'
```

SELECT * FROM PRODUCT WHERE1;

ID	NAME	PRICE
1	Clavier	40.0
2	Clavier (copie)	40.0
3	Ordinateur	540.0
4	Moniteur	80.0
6	Clavier + Ordinateur + Moniteur	660.0

(5 rows, 2 ms)

SELECT * FROM PRODUCT_SOURCES;

PRODUCT_ID	SOURCE_ID
6	1
6	3
6	4

(3 rows, 4 ms)